UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TRABAJO FIN DE GRADO:

Nivelador automático para trabajos de construcción

Autor: Iván Cerdá Andrés

Tutor: Dr. Carlos Pascual Domínguez Montagud

ETSID (UPV), Valencia, Septiembre del 2016

Tabla de contenido

1. OBJETO DEL PROYECTO

El presente proyecto es un Trabajo de Final de Grado (TFG) del Grado en Ingeniería Electrónica Industrial y Automática, de la mención de Informática Industrial, cursado en la Escuela Técnica Superior de Ingeniería del Diseño de la Universidad Politécnica de Valencia.

El objetivo del presente trabajo es el estudio, desarrollo e implementación de un sistema de control para la nivelación automática en un eje, de una estructura ante inclinaciones comprendidas entre -40º y 40º con aplicación en el ámbito de la construcción. Para conseguir dicho objetivo, el proyecto se ha desarrollado con unas herramientas de desarrollo económicas: una tarjeta STM32F3 Discovery y su microcontrolador, ARM Cortex M4, y el entorno de programación Keil 5.

Dicho sistema se basa en la utilización de un giróscopo y un acelerómetro, integrados ambos en la tarjeta STM32F3, para medir la inclinación de la estructura mediante un filtro de Kalman, y la posterior corrección de ángulo para conseguir un nivel completo, todo desarrollado en el software programado.

Para la implementación física del sistema se ha utilizado un brazo robot con movimiento en el eje “Y”, y capaz de cubrir un amplio rango de grados tanto positivos como negativos.

2. ESTADO DEL ARTE

Este proyecto nace de la idea de adaptar los métodos y avances tecnológicos al ámbito de la construcción a baja escala. Actualmente los operarios de la construcción de viviendas e interiores utilizan métodos de nivelación clásicos, como un nivelador de burbuja. Además, se valen de calzas o cuñas para conseguir un nivel correcto.

Ante estos antecedentes ha surgido la idea de desarrollar un prototipo de nivelador automático en único eje, de bajo coste y con una buena precisión, basado en el microcontrolador “STM32F3”.

A parte de la funcionalidad principal, este prototipo tiene un amplio espectro de utilidades ya que consiste en una herramienta económica, resistente y ligera, capaz de nivelarse automáticamente ante inclinaciones comprendidas entre ±40º.

3. FUNDAMENTOS TEÓRICOS

3.1. Análisis de movimiento

El presente proyecto está fundamentado en el posicionamiento de una plataforma que ejecuta el movimiento denominado de cabeceo, el mismo que será medido por medio de sensores inerciales.

3.1.1. Sistema de coordenadas

El sistema de coordenadas que se utilizará como referencia para el desarrollo de la plataforma de simulación de movimiento en base a la medición del ángulo de cabeceo, es el llamado BCS (Body Coordinate System) o sistema de coordenadas sólidas, cuyo origen se encuentra en el centro de masa del cuerpo analizado.

Este sistema se usa con frecuencia en plataformas "Strapdown", es decir, cuando los ejes de los sensores y del cuerpo donde se encuentran se mueven de forma conjunta.

3.1.2. Movimiento de Cabeceo, alabeo y guiñada

Tomando como referencia el sistema con sus tres ejes fijos en la plataforma, se denominará eje de guiñada (yaw) al eje Z, de cabeceo (pitch) al eje Y, y de alabeo (roll) al eje X, en base a los cuales se establecen tres rotaciones intrínsecas principales, es decir, relativas al sistema:

Cabeceo: inclinación o rotación alrededor del eje Y. Puede ser en sentido positivo o negativo.

Alabeo: inclinación o rotación alrededor del eje X. De igual manera puede darse el sentido positivo o negativo

Guiñada: rotación respecto de un eje vertical Z.

3.2. Sensores

Los sensores inerciales son utilizados para el estudio y análisis del movimiento, en base a las variables de aceleración y velocidad angular que son obtenidas mediante acelerómetros y giróscopos respectivamente.

3.2.1. Tecnología MEMs

Las siglas “MEMs” son en conjunto, el acrónimo de Sistemas Micro-ElectroMecánicos (MicroElectroMechanical Systems). Son definidos típicamente como dispositivos de pequeñas dimensiones compuestos por elementos activos y pasivos, micro fabricados y que realizan diferentes funciones como percepción, procesado de datos, comunicación y actuación sobre el entorno.

Gracias a los avances en el campo de los semiconductores, los MEMs son una tecnología que puede aplicarse utilizando una gran diversidad de materiales y técnicas de fabricación.

Los MEMs poseen una serie de ventajas frente a sistemas de mayor tamaño, cómo, por ejemplo:

- Posibilidad de fabricación masiva de bajo coste
- Componentes más sensibles
- Tamaño y peso reducidos
- Consumo de energía pequeño
- Alta precisión y biocompatibilidad
- Partes mecánicas específicamente diseñadas, por lo que serán más rápidas y eficientes.
- Materiales con propiedades que les permiten ser más fuertes y ligeros.
- Desarrollo de componentes electrónicos más rápidos.
- Sistemas mecánicos y ópticos más rápidos.

Dichas ventajas han favorecido su introducción en un gran número de aplicaciones y mercados, y silenciosamente han ocupado un lugar en nuestra vida cotidiana.

En este proyecto se trabaja con la tarjeta STM32F3, y concretamente con sus dos sensores principales: el acelerómetro y el giróscopo, ambos sensores de tecnología MEMs.

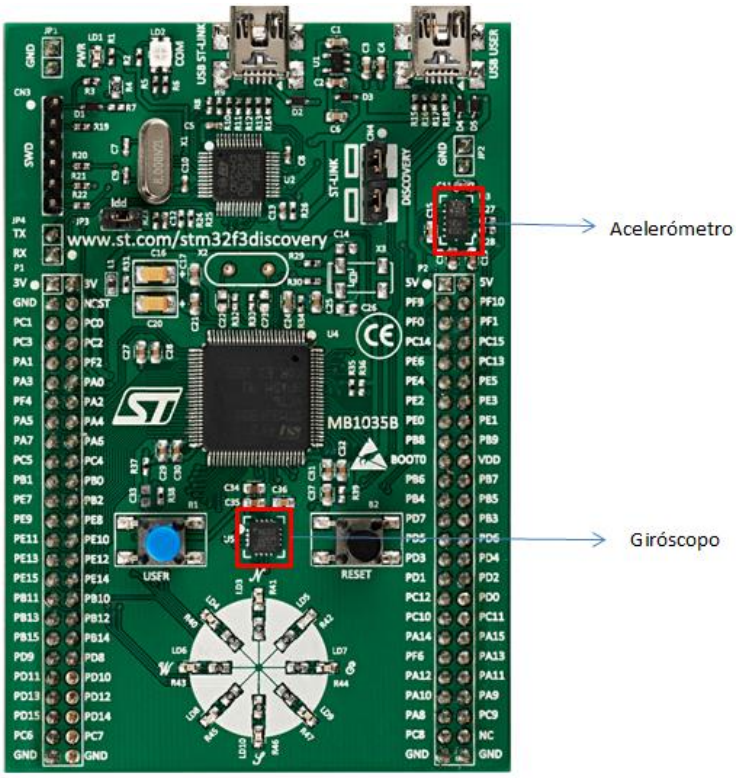


Ilustración 1- STM32F3 Discovery mostrando sensores MEMs

3.2.2. Error de deriva

El concepto de error se define como la diferencia entre el valor verdadero de una medición y el valor registrado de una medición. Hay muchas fuentes de error y éste puede ser aleatorio o sistemático.

En particular, el error de deriva o, simplemente deriva, se presenta debido a la acumulación de errores sistemáticos que son integrados en el tiempo, ya que, como se explicará en el apartado 3.3.2, el ángulo final se obtiene sumando todos los ángulos anteriores. Por tanto, si en cada una de las medidas se obtiene un error, éste se irá incrementando con el transcurso del tiempo.

3.2.2.1. El error de Bias

El error de bias es un error sistemático que aparece en las medidas realizadas con sensores y es propio del dispositivo, es decir, depende enteramente de las características de construcción del sensor. Este error se puede apreciar cuando el sensor se encuentra en estado estático, y aunque el parámetro bias viene especificado en la hoja de características del fabricante, es importante medir el valor exacto de este error al comienzo de una aplicación con tal de contrarrestar su efecto en las medidas futuras.

3.2.3. Giróscopo

El giróscopo es un dispositivo que permite conocer como varía un ángulo en el tiempo mientras se encuentre rotando, es decir, permite conocer la velocidad angular.

Inicialmente solo se utilizaban para aplicaciones militares debido a su elevado tamaño y peso, sin embargo, actualmente y gracias a la tecnología MEMs, tienen una gran versatilidad y se utilizan en numerosas aplicaciones como por ejemplo en sistemas de navegación inercial (INS), estabilización de plataformas…

3.2.3.1. Principio de funcionamiento

En los giróscopos existen 3 principios de funcionamiento principales:

- **Rotatorios**: se utiliza una masa rotando sobre un eje sostenido por uno o varios cardanes dependiendo de los grados de libertad que se deseen. Cuando se produce un movimiento en el sistema externo es posible observar un cambio en el ángulo.
- **Vibratorios**: Se caracterizan por disponer de un elemento vibrante que al ser forzado a rotar es afectado por una fuerza de Coriolis que induce vibraciones secundarias ortogonales a la vibración original. La velocidad angular se obtiene en base a dichas vibraciones.
- **Ópticos**: usan el efecto Sagnac para detectar la rotación a la cual están sometidos. Cuando dos rayos de luz circulan en direcciones opuestas dentro de un camino cerrado, el haz que circula en la misma dirección de la rotación tarda más tiempo en viajar que el haz de luz que va en sentido contrario, con lo cual el ángulo se puede obtener en base a la diferencia de camino que ven los dos rayos.

3.2.3.2. Señal entregada

Como se ha explicado en el apartado 3.2.3, la señal entregada por un giróscopo es la velocidad angular con la que está rotando el cuerpo en el que se encuentra el sensor. Por tanto, la información recibida estará en º/s o en rad/s.

En este caso el sensor cuenta con una sensibilidad de mdps/dígito, es decir, una sensibilidad de miligrados por segundo.

3.2.3.3. Características L3GD20

El L3GD20 es un sensor triaxial de velocidad angular de bajo consumo. Está compuesto por un circuito sensor y un circuito de interfaz que permite la comunicación mediante protocolos I2C o SPI.

El L3GD20 tiene una escala de ±250/±500/ ±2000 dps (grados por segundo) y es capaz de medir velocidades con un rango ajustable.

- Fondo de escala ajustable en tres parámetros: 250/500/2000 dps.
- Salida digital mediante I2C/SPI.
- 8-bit de salida.
- Dos salidas digitales (Interrupt y DataReady).
- Filtros paso bajo y paso alto integrados, con ancho de banda ajustable.
- Amplio rango de alimentación: de 2.4 a 3.6 Voltios.
- Modos de Bajo consumo y "sleep mode".
- Funcionamiento correcto en un amplio rango de temperaturas: de -40ºC hasta aproximadamente +85ºC.

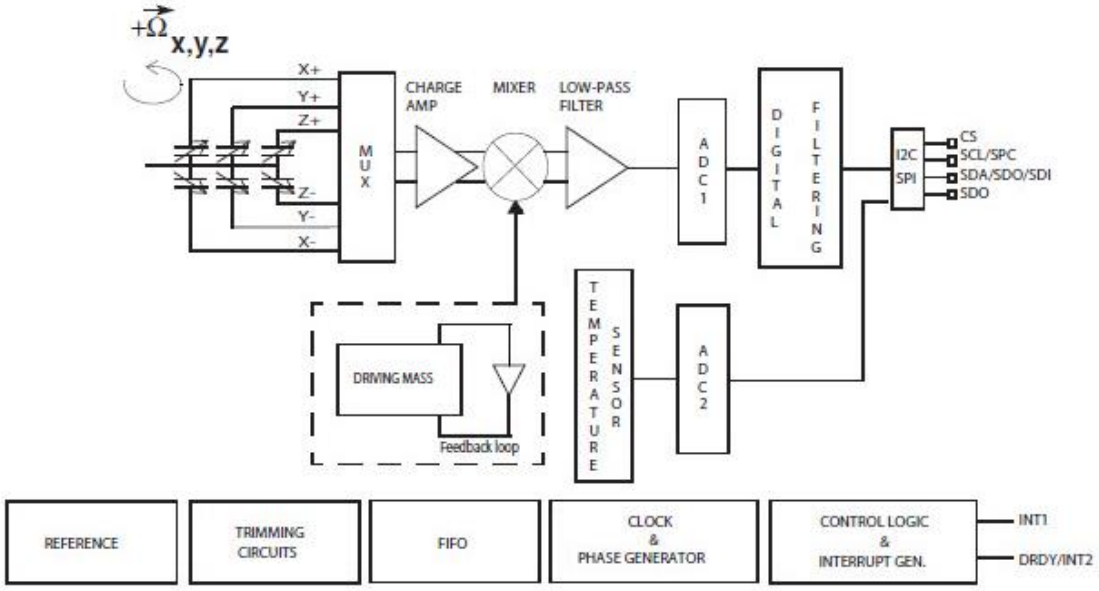


Ilustración 2 - Diagrama de bloques del L3GD20

3.2.4. Acelerómetro

El acelerómetro es un sensor que permite conocer la aceleración que se produce al realizar un movimiento a lo largo del eje en el cual se esté trabajando. Hay que tener en cuenta que este sensor mide la magnitud de aceleración en general, incluyendo en esta medida el valor de la aceleración de la gravedad.

3.2.4.1. Señal entregada

Tal y como se ha explicado en el apartado anterior, un acelerómetro mide la aceleración que sufren cada uno de sus ejes. Esta información se mide en mg o g dependiendo del sensor utilizado. En el caso de este proyecto el acelerómetro utilizado tiene una sensibilidad de mg/LSB.

3.2.4.2. Características LSM303DLH

El integrado LSM303DLHC está compuesto por dos sensores, un acelerómetro y un magnetómetro.

- El LSM303DLHC tiene unas escalas de aceleración lineal de ±(2g/4g/8g/16g), y un rango de escala magnética comprendido entre ±1.3 y ±8.1 Gauss.
- Alimentación analógica comprendida entre 2.16V y 3.6V.
- Correcta funcionalidad en un rango de temperaturas de entre -40 hasta 85ºC.
- 16 bit de salida.
- Incluye un Bus de comunicaciones en serie I2C que incluye un modo estándar, a una velocidad de 100 kHz, y otro de mayor velocidad, a 400 kHz.

A continuación, se adjunta el diagrama de bloques del integrado.

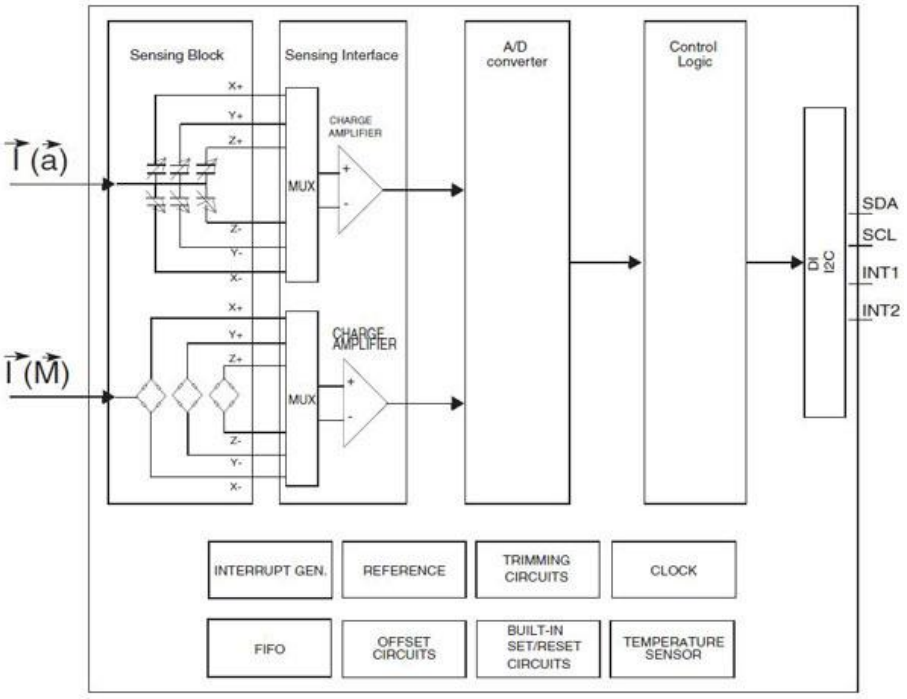


Ilustración 3 - Diagrama de bloques del LSM303DLHC

3.3. Métodos para obtener el ángulo de inclinación

En base a los sensores inerciales propuestos y sus respectivas características, a continuación se presentarán 3 métodos para calcular el ángulo de inclinación.

3.3.1. Inclinación usando solo el acelerómetro

Como se ha comentado anteriormente, el acelerómetro mide la aceleración a la cual está sometido el sensor en cada uno de sus ejes.

Hay que tener en cuenta que la sensibilidad de los acelerómetros para la medición de la inclinación aumenta cuando sus ejes son perpendiculares a la aceleración de la gravedad, es decir, paralelos a la superficie terrestre.

Primero hay que establecer que el acelerómetro se encuentra unido firmemente a la plataforma, por lo que cuando ésta se encuentra paralela a la superficie terrestre, los ejes X e Y también estarán paralelos a tierra, es decir, perpendiculares a la gravedad.

Cuando la plataforma toma una inclinación como la de la Ilustración 4, el sensor se mueve junto con la plataforma provocando que el eje Y (cabeceo) entregue una señal, debido a la componente de la gravedad presente.

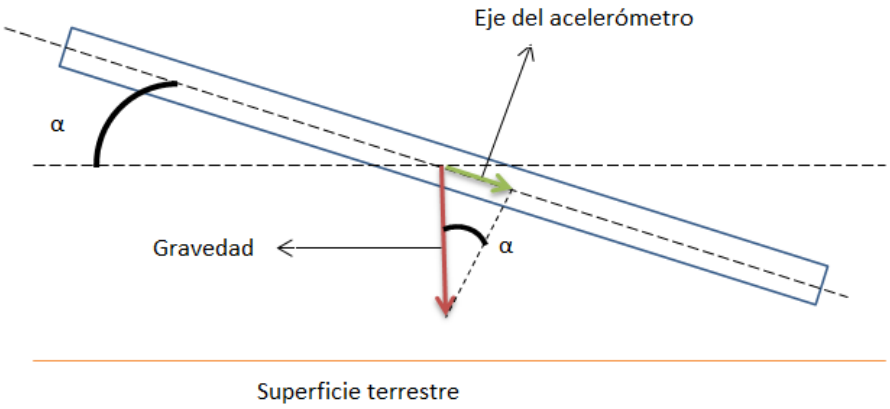


Ilustración 4 - Cabeceo con ángulo α en sentido horario

En el gráfico, la flecha verde representa lo que el acelerómetro está midiendo a causa de la gravedad y la flecha roja la aceleración de la gravedad.

Para obtener el valor del ángulo de inclinación en radianes seguiremos la siguiente fórmula:

$$PITCH = \sin^{-1}\left(\frac{A_x}{g}\right) \quad (1)$$

Hay que tener en cuenta que la aceleración entregada por el acelerómetro solo servirá para el cálculo del ángulo en condiciones en las que no se esté produciendo un movimiento que implique presencia de aceleración lineal en cualquiera de sus ejes, es decir, en condiciones en que la aceleración sea 0. Por tanto, un ligero cambio en la aceleración sobre el sensor o la presencia de vibraciones producirá que el ángulo que se obtiene no sea el correcto, con lo cual, el utilizar únicamente el acelerómetro como sensor de medición no resulta viable si el sistema presenta aceleración lineal.

3.3.2. Inclinación usando solo el giróscopo

Como se ha comentado en el apartado 3.2.3 el giróscopo entrega como dato la velocidad angular. Para obtener el ángulo de inclinación es necesario integrar este dato, teniendo en cuenta como concepto de integración, multiplicar la velocidad angular por un tiempo determinado "dt".

La constante de tiempo que se utiliza depende únicamente de las necesidades que se tengan, así como también de cómo se vaya a procesar la señal. Si los intervalos de tiempo son muy pequeños se vuelve necesaria la utilización de dispositivos con una alta velocidad para el procesamiento de la información.

$$\frac{d(\acute{a}ngulo)}{dt} = velocidad\ angular = se\tilde{n}al\ gir\acute{o}scopo \quad (2)$$

$$d(\acute{a}ngulo) = velocidad\ angular * dt \quad (3)$$

$$\int velocidad\ angular * dt = \int se\tilde{n}al\ del\ gir\acute{o}scopo * dt = \acute{a}ngulo \quad (4)$$

Viendo las fórmulas anteriores se puede observar que, a través del producto de la velocidad angular por el tiempo en el que se está realizando el movimiento, se puede obtener una pequeña variación en el ángulo.

Sin embargo, este modelo solo muestra un ángulo en un instante determinado. Para encontrar el ángulo de inclinación, es necesario un algoritmo acumulativo en el cual el resultado sería este dato. El modelo para obtener el ángulo de inclinación sería:

$$\acute{a}ngulo_k = \acute{a}ngulo_{k-1} + u_k * dt \quad (5)$$

Siendo u_k la señal acondicionada del giróscopo, dt el tiempo de muestro, $ángulo_k$ el ángulo presente y $ángulo_{k-1}$ el ángulo anterior.

De este modelo encontramos 2 inconvenientes:

- Sólo se obtendrá valores fiables en intervalos muy cortos de tiempo, ya que al tratarse de un modelo acumulativo el error se irá acumulando a lo largo del tiempo lo que provocará que en intervalos largos de tiempo los valores obtenidos estén muy lejos de la medida real.
- Es necesario un estado inicial o ángulo de partida, lo cual es muy complicado a menos que el sistema siempre tenga un estado inicial conocido.

Teniendo en cuenta estos dos inconvenientes se puede deducir que es muy difícil conseguir una medida fiable del ángulo de inclinación solo mediante el uso del giróscopo, por lo tanto, es necesario la utilización de otro dispositivo que contrarreste en cierto modo los problemas del uso del giróscopo.

3.3.3. Inclinación usando el acelerómetro y el giróscopo

Teniendo en cuenta los dos métodos explicados en los apartados anteriores se puede obtener que el uso de los sensores de forma individual lleva inconvenientes implícitos que conducen a que el valor del ángulo de inclinación obtenido sea erróneo.

Con tal de solucionar estos problemas surge la necesidad de trabajar con ambos sensores de forma simultánea, con el fin de obtener las mejores características de cada uno de ellos y contrarrestar los errores que incluyen al sistema.

3.4. Tratamiento de los datos

Con tal de poder lograr la fusión sensorial y conseguir un nivelador fiable, es decir, que amplifique las características positivas de cada sensor y atenúe los problemas individuales hay que utilizar algún tipo de filtro. Los dos más utilizados son el “filtro de Kalman” y el “filtro complementario”.

3.4.1. Filtro complementario

Los filtros complementarios son una solución determinista del problema de estimar un conjunto de variables a partir de mediciones provenientes de diferentes sensores, planteada en el dominio frecuencial.

En particular, cuando se desea estimar un ángulo a partir de los datos provenientes de un acelerómetro y un giróscopo, se diseña un filtro complementario para aprovechar los datos dentro del rango de frecuencias donde el ruido y las perturbaciones de los sensores son menores.

Al pasar los datos del acelerómetro por un filtro paso-bajo se eliminan las perturbaciones de alta frecuencia y se aprovecha la precisión de los resultados a largo plazo. De forma análoga, cuando se filtra la señal del giróscopo con un filtro paso-alto se eliminan los efectos negativos de la acumulación del error y se aprovecha su inmunidad a las vibraciones y otras aceleraciones externas que perturban las estimaciones provenientes del acelerómetro. Por lo tanto, el esquema de fusión sensorial basado en un filtro complementario es el siguiente:

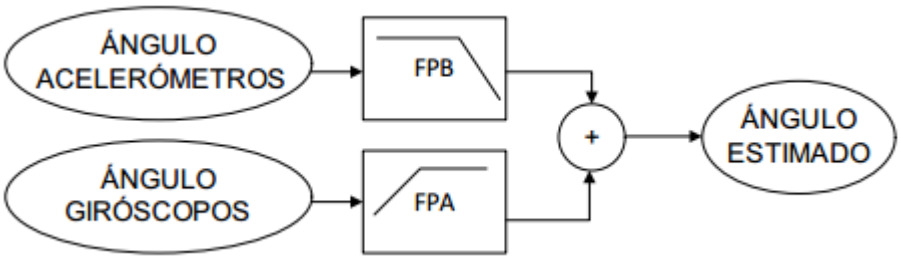


Diagrama de Bloques 1 - Fusión sensorial filtro complementario

3.4.2. Filtro de Kalman

El Filtro de Kalman es un algoritmo que fue desarrollado por Rudolf E. Kalman en 1960 y describe una solución recursiva para problemas de filtrado de datos discretos. Desde su publicación, este algoritmo ha sido objeto de un exhaustivo estudio debido a las enormes aplicaciones que puede ofrecer, especialmente en sistemas, autónomos o asistidos, de navegación.

El Filtro de Kalman es un estimador óptimo que puede implementarse de manera sencilla en sistemas de carácter tanto lineal como no lineal, y cuyo procesamiento de datos es de carácter recursivo. Se denomina óptimo ya que recibe y procesa todas las mediciones disponibles y en base a estas, estima el valor actual de las variables de interés. Con el fin de llevar a cabo este algoritmo es necesario [17]:

a) Conocimiento del sistema en el cual va a ser implementado el filtro, así como también mediciones dinámicas provenientes de los dispositivos a utilizarse.

b) La descripción estática del ruido presente en el sistema, la información acerca del error, la incertidumbre en el modelo.

c) Conocer las condiciones iniciales de las variables más importantes presentes en el modelo.

La palabra “recursivo” está relacionada con un algoritmo en el cual el filtro no tiene la necesidad de mantener almacenados todos los datos anteriores de las variables, por lo tanto no tiene que reprocesarlos cada vez que una nueva medida es tomada. En lugar de esto hace uso de estados anteriores, es decir, del último valor calculado a partir del cual, y junto con las nuevas medidas tomadas, el algoritmo entrega nuevos resultados que posteriormente serán considerados como estados anteriores. Con lo cual, la implementación de este filtro resulta muy útil en sistemas de tiempo real.

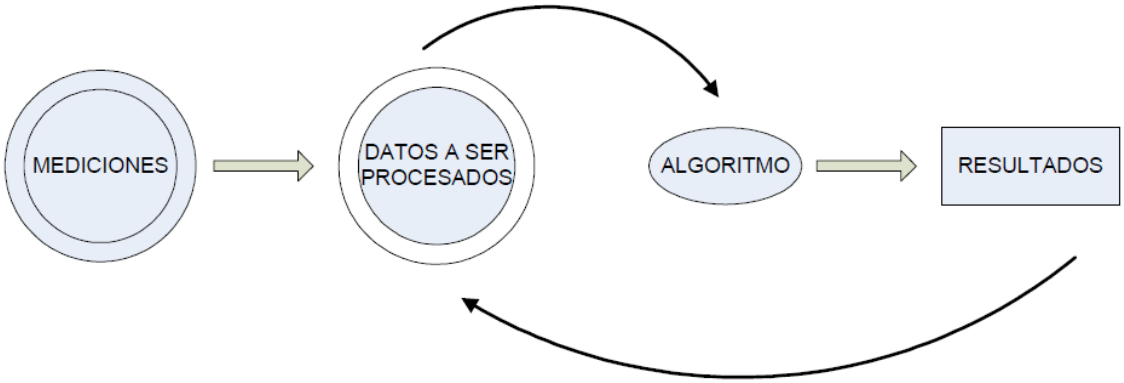


Ilustración 5- Esquema general del algoritmo de Kalman

Estas características hacen del filtro Kalman una poderosa herramienta para la estimación de estados pasados, presentes y futuros, e inclusive puede trabajar en circunstancias en las que la naturaleza precisa del modelo del sistema resulte desconocida.

3.4.2.1. Filtro Discreto de Kalman

El Filtro Discreto de Kalman es un algoritmo mediante el cual se realiza un proceso de predicción y otro de corrección mediante la medición y observación de un grupo de variables presentes en el sistema a tratar. Dicho conjunto de variables forma el vector de estados y el observador. Estas variables se encuentran representadas en las ecuaciones del sistema dinámico del fenómeno físico en estudio, basadas en el efecto del ruido presente en las observaciones, así como en la incertidumbre de la dinámica del sistema

Con el fin de implementar el filtro de Kalman para remover el ruido presente en la señal que se está midiendo en un proceso, se debe tener en cuenta que la dinámica del sistema se tiene que presentar como un modelo de tipo lineal. Aunque también se ha desarrollado el algoritmo para sistemas no lineales, este tipo de desarrollo no se va a tratar en este trabajo.

Como ya se mencionó, el filtro de Kalman usa un algoritmo de carácter recursivo con el fin de obtener una variable estimada que se acerque lo más posible a la realidad.

A continuación se presentará el modelo general del sistema de la variable a ser estimada, así como también la ecuación del observador.

3.4.2.2. Modelo

El sistema dinámico en el cual se desea aplicar el filtro de Kalman debe expresarse como un modelo de carácter lineal, en el cual se trata de estimar el estado x ϵR^n. Dicho sistema se puede describir de manera general mediante la siguiente ecuación de estado:

$$x_k = Ax_{k-1} + Bu_k + w_{k-1} \quad (6)$$

Con una medición z ϵ R^m :

$$z_k = Hx_k + v_k \quad (7)$$

Dónde:

x ---> Estado del sistema.

k ---> Tiempo.

z ---> Valor observado.

u ---> Entrada del sistema.

v ---> Ruido en la medición.

w --->Ruido en el proceso.

A, B, H --->Matrices determinísticas que definen la dinámica del sistema.

La matriz A de dimensiones (n x n), en la ecuación describe la relación que existe entre el estado en el momento *K -1* con el estado actual del instante *K*. La Matriz B de dimensiones (m x l), relaciona la entrada de control u ϵ R^l con el estado *x*. La matriz H de dimensiones (m x n) presente en la ecuación de medición, relaciona el estado *x* con la observación z_k.

Cabe aclarar que, tanto la matriz A como la matriz H pueden cambiar en cada paso de tiempo o medición, pero si bien esto es posible, también es completamente válido que estas puedan permanecer constantes en las ecuaciones en todo el tiempo.

Los vectores *w* y *v* son independientes el uno del otro y representan el ruido gaussiano blanco con media cero, presente en el proceso y en las observaciones respectivamente.

Los vectores *w* y *v* además traen consigo asociadas las matrices de covarianza *Q* y *R* que en general son diagonales, pudiendo también no serlo.

En general, estas medidas seguirán una distribución de probabilidad Normal, de media nula y matriz de covarianza Q, para *w* y R para la covarianza de *v*.

$$p(w) \sim N(0,Q)$$
$$p(v) \sim N(0,R)$$

En la práctica, la matriz de covarianza de la perturbación del proceso *Q* y la matriz de covarianza de la perturbación de la observación *R* pueden variar en el tiempo, pero en general y por facilidad, también pueden ser consideradas como constantes. Veamos esto con más detalle.

3.4.2.3. Puesta a punto del filtro de Kalman

Como se señaló en las ecuaciones, tanto el proceso a ser estimado como las mediciones del mismo están sujetos a variaciones aleatorias de distribución gaussiana, las cuales se representan con las variables aleatorias (w_k) y (v_k) de varianzas (σ^2_w) y (σ^2_v). R. Kalman demostró que éste presenta un funcionamiento óptimo cuando los parámetros de ajuste (Q) y (R) son igual a la covarianza del proceso (σ^2_w) y la covarianza de las mediciones (σ^2_v) respectivamente

$Q=\sigma^2_w$. w_k -----> N(0,Q)

$R=\sigma^2_v$ v_k ------> N(0,R)

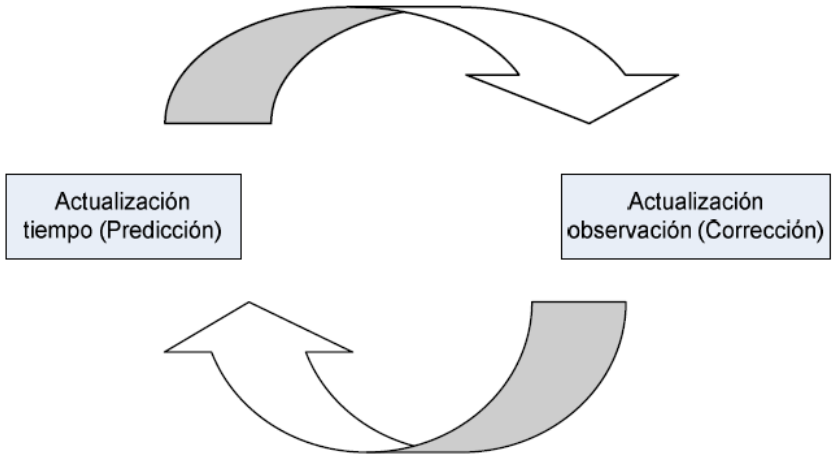
Para obtener la covarianza de las mediciones basta con tomar un conjunto de mediciones *off-line* y realizar un análisis estadístico de ellas para aproximarla. En cuanto a la covarianza del proceso, no existe un procedimiento estándar para estimarla ya que no se tiene acceso directo al proceso, sino a sus mediciones. Por lo general se seleccionan valores pequeños cuando se confía mucho en el modelo desarrollado, y valores más altos a medida que se tienen mayores dudas sobre él. Finalmente, se puede lograr un buen funcionamiento si se seleccionan los parámetros en base a un procedimiento de prueba y error.

3.4.2.4. Algoritmo

El algoritmo para el filtro de Kalman puede dividirse en dos grupos de ecuaciones a utilizarse: las primeras son las ecuaciones que se actualizan en el tiempo o también llamadas de predicción, y el segundo grupo se refiere a las ecuaciones de actualización mediante observaciones conocidas también como de corrección.

Las ecuaciones de predicción son las encargadas de obtener la estimación a priori de la matriz de covarianza del error, así como del estado actual en el tiempo K, en base al estado anterior en el tiempo K-1. Las ecuaciones de corrección en cambio, tienen como objetivo realizar una retroalimentación, es decir, la incorporación de nuevas mediciones (información) en la estimación a priori del estado, con el fin de conseguir una estimación a posteriori mejorada.

De ahí que, para conseguir una estimación final del estado, sea necesaria la utilización de la etapa de predicción y de la de corrección en el algoritmo del filtro de Kalman



.

Ilustración 6 - Ciclo del Filtro Discreto de Kalman

Las ecuaciones específicas para las etapas de predicción y corrección se muestran en la Tabla 1 y en la Tabla 2 respectivamente.

$\hat{x}_k = A\hat{x}_{k-1} + Bu_k$	(8)
$P_k^- = AP_{(k-1)}A^T + Q$	(9)

Tabla 1- Ecuaciones de predicción para el Filtro discreto de Kalman

En la Tabla 1 se puede apreciar cómo se realiza la actualización del estado y de la matriz de covarianza del error, llevándolos desde el instante K-1 hasta el momento actual K. Las matrices A, B y Q se han comentado anteriormente.

$K_k = P_k^- H^T (H P_k^- H^T + R)^{-1}$	(10)
$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-)$	(11)
$P_k = (I - K_k H) P_k^-$	(12)

Tabla 2 - Ecuaciones de corrección para el Filtro discreto de Kalman

La Tabla 2 muestra las ecuaciones correspondientes a la etapa de corrección.

La primera acción a tomarse en la etapa de corrección es realizar el cálculo de la llamada ganancia de Kalman, K_k en (10). Este es un factor de ponderación cuyo objetivo es minimizar la covarianza del error de la nueva estimación del estado. La siguiente tarea consiste en tomar una medición del proceso (observación) en el momento *K*, obteniendo así el valor de la variable Z_k. Esta medida actualizada del proceso permite obtener una estimación a posteriori del estado en (11), es decir se consigue mejorar la estimación a priori conseguida en la etapa de predicción.

Como tarea final en las ecuaciones de corrección se procede al cálculo de la estimación a posteriori de la matriz de covarianza del error mediante (12).

Después de cada par de actualizaciones (predicción y corrección), se puede apreciar claramente como una variable estimada con anterioridad puede ser mejorada por medio de una observación, y al mismo tiempo, este nuevo estado estimado es usado para dar inicio una vez más a la etapa de predicción. De esta manera el filtro de Kalman cumple con su característica claramente identificada de recursividad.

A continuación se presenta en la Ilustración 7, el algoritmo completo del Filtro de Kalman combinando las ecuaciones de la Tabla 1 y Tabla 2.

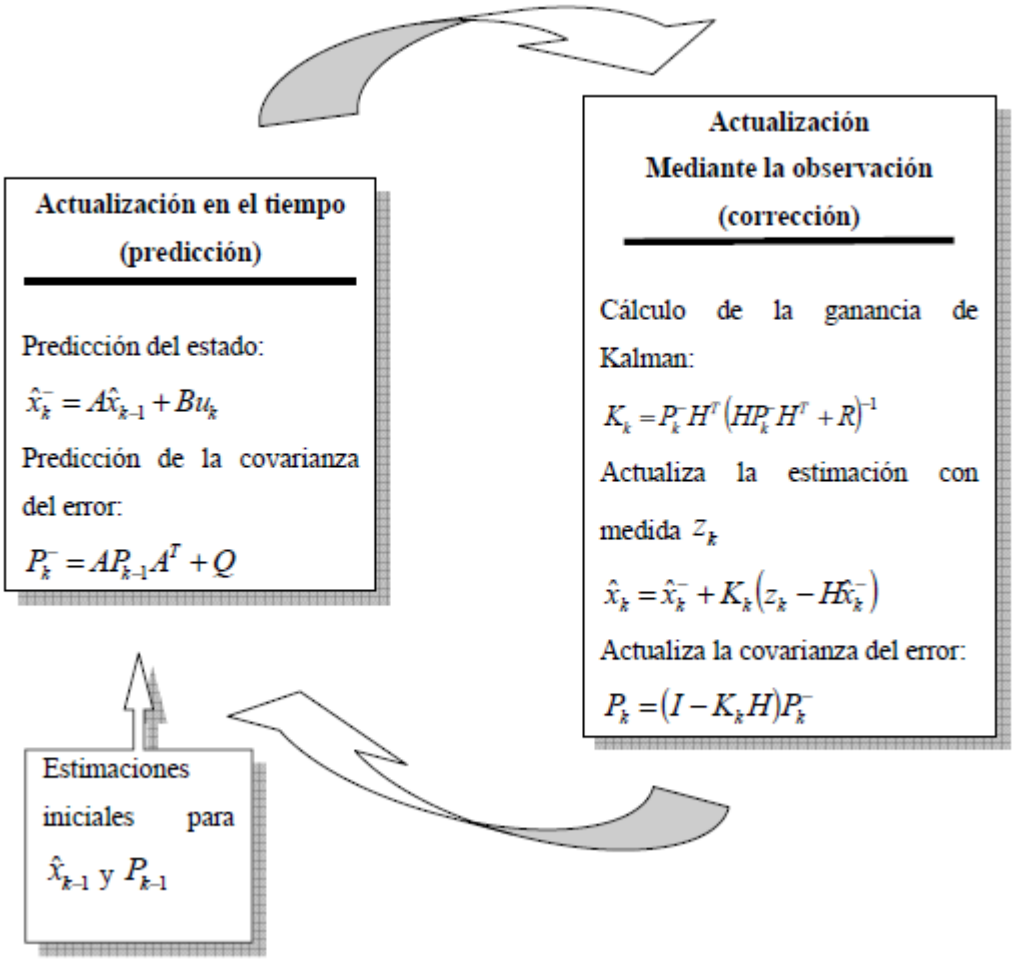


Ilustración 7 - Algoritmo completo de la operación del Filtro de Kalman, tomado de [11]

En el caso de la matriz de la covarianza, la estimación inicial con la cual dará inicio el filtro de Kalman puede ser la matriz identidad, mientras que en el caso de la estimación inicial para el estado podría asignarse cualquier valor a conveniencia del proceso en donde se esté aplicando el filtro.

3.4.2.5. Parámetros del filtro y sintonización

Como se observó en el grupo de ecuaciones que forman el filtro de Kalman, existen dos parámetros que son de gran importancia ya que representan las matrices de covarianza de las perturbaciones en el proceso y en las observaciones Q y R respectivamente.

En el caso del parámetro *R*, este puede ser obtenido vía *off-line* (fuera del sistema), es decir, antes de ser implementado en el filtro, con lo cual se podría encontrar su valor de manera práctica mediante la toma de muestras de mediciones y, así, hallar la varianza en el ruido o perturbación presente en las observaciones.

Cabe indicar también que si bien el parámetro *R* puede ser calculado experimentalmente, también podría asignarse por ensayo y error dependiendo de cómo responda el sistema.

Sea cual fuese la manera como se decida establecer el parámetro *R* en el filtro de Kalman, siempre hay la posibilidad de manipularlo para obtener un valor ideal para el proceso, para esto se tiene ciertas pautas a seguir:

Si las perturbaciones en las observaciones son grandes, entonces *R* será grande, por tanto, si se observan las ecuaciones de la ganancia de Kalman se tiene que K_k será pequeña, lo cual significa que no se dará mucha credibilidad a la medición (observación) en el momento del cálculo del siguiente $\hat{x}_k$.

Por otra parte, si en el momento de manipular la variable R, a esta se le da un valor pequeño, es decir, se considera que las perturbaciones en la observación son pequeñas, la ganancia de Kalman K_k será grande y se estará dando mayor peso o credibilidad a las mediciones (observaciones) cuando se calcule el siguiente $\hat{x}_k$.

La determinación del parámetro Q es un poco más complicada. Esto es debido a que es la representación de ruido en el proceso, y por lo general, esto es muy difícil de conocer ya que se necesitaría poder observar, de manera directa, el proceso que se está estimando.

En ciertas ocasiones, procesos relativamente sencillos pueden dar buenos resultados con tan solo la inyección de suficiente incertidumbre en su modelo por medio de la selección del parámetro Q, y de esta manera se esperaría que los datos arrojados por el proceso sean confiables.

Por otra parte, si los parámetros Q y R son considerados constantes y los valores que se les fueron asignados son los correctos, lo que se obtendrá finalmente como resultado es que tanto la matriz de covarianza del error $P_{k,}$ así como la ganancia de Kalman K_k se estabilizarán rápidamente y llegarán incluso a permanecer constantes en el proceso.

Nótese además que, en el caso del sistema propuesto las matrices de covarianza, R y Q son diagonales debido a que se asume que no existe correlación alguna entre las mediciones realizadas para las variables de estado. Si se conociera que una de las variables tiene influencia sobre la otra, entonces las matrices dejarían de ser diagonales y los elementos con “0” pasarían a tener valores dependientes de la covarianza entre las variables de estado. Q es una matriz que puede ser encontrada en base al modelamiento

del comportamiento del sistema en estudio. Normalmente es muy complicada de conseguir y muchas veces es colocada entre rangos ya establecidos previamente. Un criterio válido para colocar valores en la matriz Q sería en base a la suposición de que el sistema posee medidas muy fiables, caso en el cual los valores de Q serán pequeños. En caso contrario, si se conoce que el proceso es poco fiable, entonces Q contendrá valores altos. El término w_{k-1} en (5), en muchas ocasiones es despreciado en el momento de la implementación práctica ya que, al representar el ruido en el proceso o sistema modelado, éste es implícito a su comportamiento. Por otra parte, si se desea realizar una simulación del sistema modelado, el término w_{k-1} resulta de vital importancia ya que permite al algoritmo conocer que perturbaciones pueden presentarse en el sistema en la etapa de predicción, y así poder compensarlos en la etapa de corrección. Sin embargo, ya sea de manera práctica o por simulación, el término Q siempre debe estar presente en el algoritmo FK ya que interviene de manera directa en la estimación de la covarianza del error.

La Covarianza del error estimado proviene del error que puede existir entre la variable estimada y su posterior corrección, siendo esto un factor preponderante en el cálculo de la ganancia K que sirve para corregir la variable estimada. Además, nótese que esta covarianza de error es nuevamente recalculada mediante la expresión (12). Esta actualización o corrección de P_k^- para conseguir una nueva P_k implica que a lo largo del tiempo debe existir una tendencia a minimizar este parámetro, cuyo caso ideal sería 0 pero que normalmente, y en un proceso donde todas las variables han sido correctamente modeladas, puede llegar a valores muy pequeños. En muchos casos puede presentarse que P_k , además de llegar a tomar valores cercanos a cero, puede estabilizarse en un valor fijo después de varias iteraciones del algoritmo. Es entonces cuando se puede decir que el Filtro ha convergido a un valor estable de P_k . Si esto sucede, el proceso de predicción y corrección de P_k puede omitirse y las ecuaciones (9) y (12) pueden ser reemplazadas por una única constante P_k .El número de iteraciones necesarias para que se estabilice el Filtro depende, en gran medida de R. Mientras más grande sea R, mayor número de iteraciones se necesitarán para que P_k logre alcanzar un valor estable.

Por otra parte, refiriéndose al término υ_k, éste desaparece en el algoritmo presentado en la Ilustración 7 pero su acción queda representada por la matriz R, que en la expresión (10) implica una variación en el comportamiento de K, que no es más que un peso o ganancia que se da a la relación entre lo medido por el observador y lo estimado en la etapa de predicción, como se puede apreciar en la expresión (11) . Si bien R es un término que puede ser obtenido mediante un proceso de pruebas en base a medidas del observador, también es factible que éste pueda ser variado de manera heurística en base a las necesidades del comportamiento del FK, tomando como referencia que, mientras más alto sea el valor colocado en R, menor será el peso que se da al término de innovación en la ecuación (11). Viéndolo como algo práctico, podría decirse que, si R tiene un valor alto, entonces implicaría que el observador es muy poco confiable en sus medidas y por tanto se debe dar el mínimo peso posible al término de innovación, llegando al límite donde K=0, R=∞ y el factor de innovación sea ignorado completamente, caso en el cual según la expresión (11)se presente que $\hat{x}_k = \hat{x}_k^-$

3.4.3. Integración de sensores mediante el filtro de Kalman

Anteriormente se presentó el algoritmo matemático del Filtro de Kalman, pero es necesario conocer la manera de implementarlo y las aplicaciones que puede tener.

Una de las mayores aplicaciones que se le está dando al filtro de Kalman en la actualidad es en sistemas de navegación, donde se dispone de dispositivos como acelerómetros, giróscopos, GPS, cada uno de los cuales posee sus propias características que hacen que su información pueda, o no, ser confiable en ciertos aspectos.

Con esto en mente, el filtro de Kalman se encarga de utilizar las mejores características de cada uno de estos sensores para obtener una información veraz y utilizable para los sistemas de navegación.

En capítulos posteriores se explicará de mejor manera como utilizar los datos, tanto del giróscopo como del acelerómetro para que, con el uso del Filtro de Kalman, se logre integrar sus mejores características y así conseguir el dato del ángulo lo más fiable posible.

4. ESPECIFICACIONES DEL PRODUCTO A DESARROLLAR

4.1. Definición del producto

El producto a desarrollar consiste en el desarrollo de un sistema mecánico que permita medir la inclinación de una plataforma en un único eje (Eje Y de la plataforma), y nivelarla posteriormente. Su principal aplicación será el ámbito de la construcción a bajo nivel.

Este sistema estudiará tres formas de realizar estas funciones, comparando los resultados obtenidos y eligiendo el mejor para el sistema final.

Para ello en este apartado se obtendrá:

- Programación de la adquisición y tratamiento de los datos provenientes de sensores inerciales.
- Programación de la tarjeta STM32F3 Discovery para el control de motores y el algoritmo para la nivelación.
- Construcción de un prototipo de una plataforma niveladora.

4.2. Requisitos específicos

4.2.1. Especificaciones del Hardware

- Tarjeta STM32F3 Discovery:
  - Comunicación con el PC puerto USB
  - Sensores inerciales integrados con comunicación SPI/I_2C
  - Comunicación de secuencias de control de motores con pines E/S
- Prototipo de la plataforma niveladora:
  - Sistema de sujeción que permita que la tarjeta STM se mueva junto con el prototipo para una correcta medida
  - Sistema que permita el movimiento desde -40º hasta 40º rotando alrededor del eje Y (ángulo de cabeceo).

4.2.2. Especificaciones del Software

Se trabajará en el entorno de programación en Keil 5.0 y con el lenguaje de bajo nivel “C”. Se realizará un código genérico que permita en un futuro la exportación de dicho código a otras tarjetas de la familia STM mediante el uso de las librerías HAL, comunes para las tarjetas de la familia STM.

4.2.3. Operaciones a realizar

- El sistema debe medir el ángulo de inclinación de la plataforma de forma precisa.
- El sistema debe de tratar dicho ángulo para posteriormente nivelarlo.
- Se debe permitir el control del motor responsable del movimiento del prototipo para conseguir una correcta nivelación.
- Montaje del prototipo físico, con estudio y desarrollo de los sistemas de alimentación, adaptación y movilidad.

4.3. Diseño electrónico y mecánico de la plataforma

A continuación se explicarán los diferentes sistemas y sensores que se utilizarán en el presente trabajo para conseguir el objetivo de nivelar una plataforma ante ángulos comprendidos entre -40 y 40º.

4.3.1. Microcontrolador

El microcontrolador será el encargado de la adquisición y tratamiento de los datos procedentes de los sensores, además de controlar el motor del prototipo con el fin de conseguir una buena nivelación.

Con tal de optimizar y evitar el diseño e implementación de una etapa analógico-digital externa al sistema para adaptar la señal de los sensores se ha decidido utilizar una de las tarjetas de la familia STM ya que integran tanto los sensores como los convertidores ADC.

A continuación se realizará una comparación entre diferentes tarjetas de la familia STM para elegir la tarjeta óptima para el desarrollo de este proyecto.

4.3.1.1. Comparación entre microcontroladores

- ***STM32F4 Discovery***

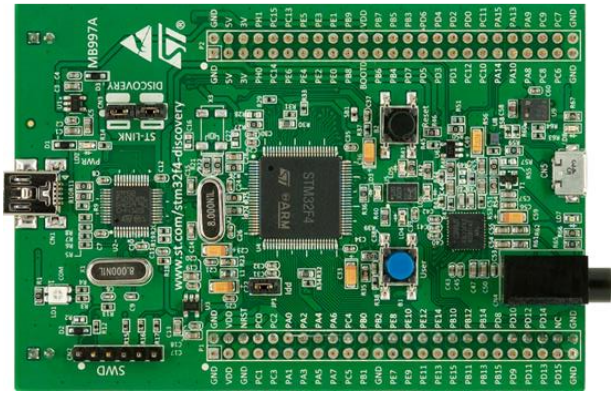


Ilustración 8 - Tarjeta STM32F4 Discovery

La tarjeta STM32F4 Discovery es una alternativa muy frecuente ya que está ampliamente documentada. Esta tarjeta cuenta con un 1 Mb de memoria Flash y 192 Kb de memoria RAM, 4 leds de señalización y dos pulsadores (uno de usuario y otro de Reset). Unas de sus características más destacables son su precio reducido de 12.90€ y la alta capacidad de configuración.

Los grandes inconvenientes de este microcontrolador son los sensores con los que cuenta. Tal y como se ha visto en el apartado 3.3, para conseguir un correcto nivelador sería necesario un acelerómetro y un giróscopo, sin embargo, esta placa únicamente cuenta con un acelerómetro. Este inconveniente lleva a descartar esta opción ya que no sería posible realizar un buen nivelador sin ambos sensores.

- ***STM32F429I Discovery***

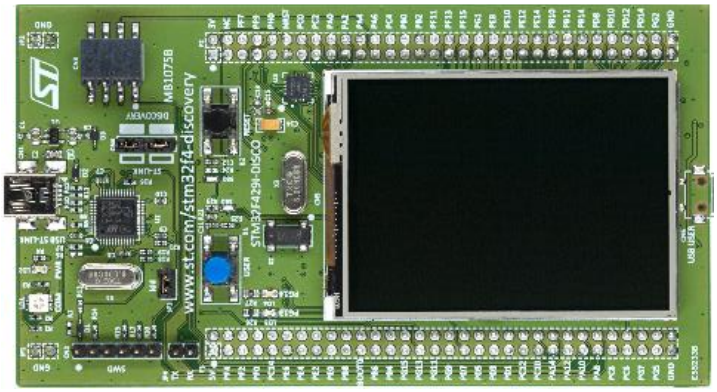


Ilustración 9 - Tarjeta STM32F429I Discovery

Esta tarjeta tiene como principal ventaja tener mayor velocidad de procesador (180 MHz) y una mayor memoria, tanto Flash (2 Mb) como RAM (256 Kb). Por tanto, si se desea realizar un proyecto con una necesidad de alta velocidad de adquisición y tratamiento de datos, o proyectos con alta necesidad de memoria esta tarjeta sería la más indicada. Además cuenta con una pantalla LCD y dos pulsadores.

Centrándonos en este proyecto, el requisito indispensable es contar con los sensores necesarios para el estudio del movimiento. Esta tarjeta cuenta con un giróscopo pero no con un acelerómetro, por tanto, tal y como ocurría con la placa anterior, queda descartada para el desarrollo del proyecto.

- ***STM32F3 DISCOVERY***

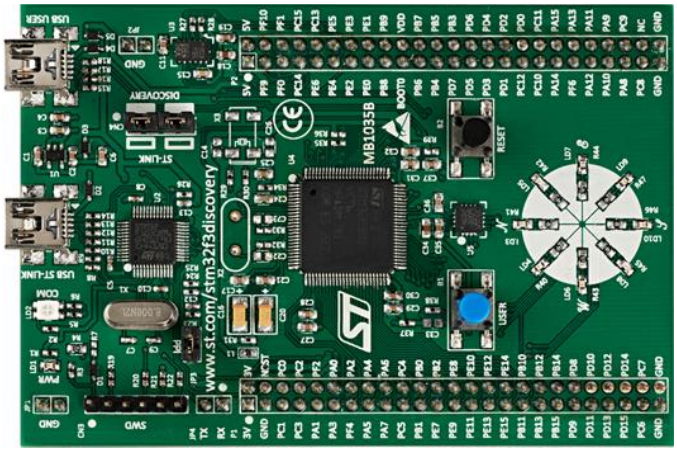


Ilustración 10 - Tarjeta STM32F3 Discovery

Este microcontrolador tiene como principal ventaja contar con los sensores necesarios para el estudio del movimiento, es decir, cuenta con un acelerómetro, un giróscopo y un magnetómetro.

Como principal inconveniente encontramos sus bajas prestaciones en comparación a las tarjetas anteriormente comentadas: velocidad de procesador de 72 MHz, memoria flash de 256 Kb y memoria RAM de 48 Kb.

Tal y como se puede observar, estos valores son mucho menores que las opciones anteriores, sin embargo, para el proyecto que se va a desarrollar estos valores no son críticos y esta opción es la única que cuenta con los sensores necesarios para el estudio del movimiento.

Teniendo en cuenta todas las características mostradas, la opción escogida para la realización del proyecto ha sido la tarjeta STM32F3 Discovery.

En la Tabla 3 se muestra un resumen de todas las características de las tres opciones estudiadas:

	Sensores	Nº pines	Alimentación	Tamaño (mm)	Mem. RAM	Mem. Flash	Precio (aprox)
STM32F4 DISCOVERY	Acelerómetro	100	3-5 V	97x66	192 KB	1MB	12,90 €
STM32F429I DISCOVERY	Giróscopo	144	3-5 V	119.3x66	256 KB	2MB	20,80 €
STM32F3 DISCOVERY	Acelerómetro y giróscopo	100	3-5 V	97x66	48 KB	256 KB	11,39 €

Tabla 3- Comparación entre Microcontroladores

4.3.1.2. Descripción de la tarjeta STM32F3 Discovery

La placa STM32F3 Discovery ofrece las siguientes características:

- Microcontrolador STM32F303VCT6 con 256 Kb de memoria Flash, 48 Kb de memoria RAM con encapsulado LQFP100
- ST-LINK/V2 incorporado con selector usar el kit como un ST-LINK/V2 independiente (con conector SWD para programación y depuración).
- Fuente de alimentación: a través del bus USB o desde una fuente de alimentación externa de 5 V.
- Sensor de movimiento ST MEMS LSM303DLHC, acelerómetro y magnetómetro de 3 ejes.
- Sensor de movimiento ST MEMS L3GD20, giróscopo de 3 ejes.
- Diez Leds: LD1 (rojo) alimentación 3.3 V, LD2 (rojo/verde) para la comunicación USB, y 8 leds de usuario LD3/10 (rojo), LD4/9 (azul), LD5/8 (naranja) y LD6/7 (verde).
- Dos pulsadores (usuario y Reset).
- USB con conector mini-B.

En cuanto al microcontrolador STM32F303VCT6 incorporado en la tarjeta se caracteriza por:

- Procesador ARM Cortex-M4 32-bits.
- FPU tiene 90 DMIPS.
- Velocidad de procesador de 72 MHz.
- 12 conversores DAC y 4 conversores ADC.
- Timers: 13 TIMs.

De todas estas características, la más significativa para el proyecto es la posibilidad de generar pulsos PWM en algunos Timers para poder controlar la velocidad del motor del prototipo, además de contar dentro del kit STM32F3 Discovery con los sensores de movimiento necesarios para la correcta medida del ángulo de inclinación en una plataforma en movimiento.

4.3.1.3. Configuración del Depurador de la tarjeta

Una parte principal de la placa STM32F3 Discovery es su depurador ST-LINK/V2. Antes de poder emplear la placa con su depurador es necesario comprobar que los jumpers están puenteados, como se indica en la siguiente imagen, y que el puente SB10 está conectado:

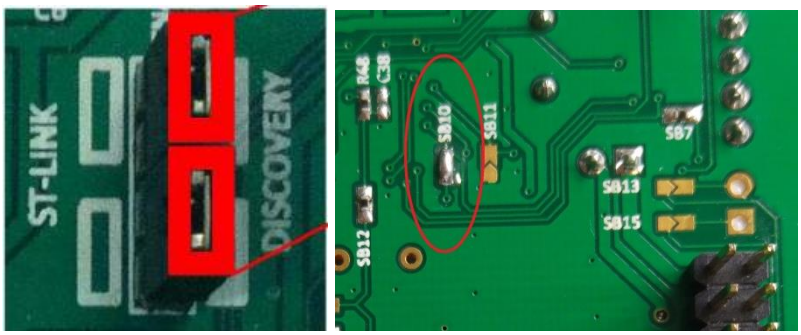


Ilustración 11 - Jumpers CN4 y puente SB10 cerrado

Para poder utilizar este depurador también hay que instalar los controladores que se pueden descargar de forma gratuita en el siguiente enlace:

Enlace descarga de Controladores para el depurador

Con tal de conseguir que se pueda utilizar la funcionalidad *printf* en la realización de los proyectos, hay que soldar el puente SB10 y ajustar el valor del Core Clock del menú de configuración del Debug al mismo valor del System Core Clock, en este caso 72 MHz.

4.4. Plataforma niveladora

La plataforma diseñada se encarga de compensar los ángulos de inclinación medidos por la tarjeta STM32F3 Discovery. Para ello se utiliza un motor de corriente continua del Kit Brazo Robótico con mando de la marca *Cebek*.

Para la estructura se han utilizado diferentes piezas “Lego Technic”, algunas piezas de plástico del kit robótico y una plataforma de la placa Make-Block para unir las diferentes piezas.

Tal y como se ve en la imagen se ha añadido un contrapeso para conseguir una mayor estabilidad en el sistema.

Hay que tener en cuenta que al estar integrados los sensores inerciales en la tarjeta, ésta debe estar incorporada y sujeta al sistema ya que es la encargada de medir el ángulo en cada momento. Para conseguir la sujeción al sistema se ha diseñado una plataforma de contrachapado en la cual, la tarjeta encaja perfectamente, evitando así sistemas de sujeción externos. A su vez, en uno de los laterales de esta plataforma se ha dejado un

espacio libre para poder comprobar, con un dispositivo externo, si el sistema está nivelado correctamente.

El sistema mecánico de la plataforma fue diseñado para que exista un rango de movimiento de 180º (-90º a 90º) para el eje Y, aunque por limitaciones de la precisión de los sensores se trabajará en un rango de entre -40 y 40 grados. Este sistema solo ejecuta este movimiento, es decir, solo nivelará el sistema alrededor del eje Y.

El sistema final con la tarjeta STM32F3 sobre el sistema es el siguiente:

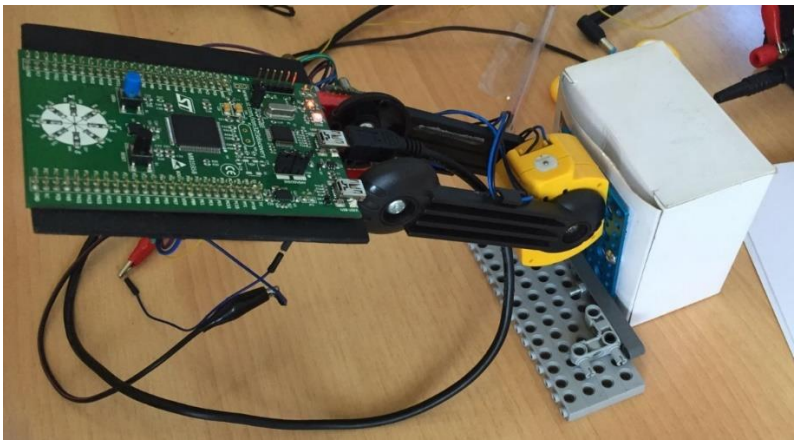


Ilustración 12 - Plataforma niveladora

A continuación se explicará con detalle cada una de las partes del sistema y el ensamblaje del prototipo.

4.4.1. Partes

Al tratarse un prototipo del nivelador automático en un eje, se decidió construirlo con materiales fácilmente accesibles y de precio reducido. Por ello se han utilizado componentes de diferentes características, como son:

- Kit brazo robot C-9895 (Cebek): este kit consiste en un brazo robot de 5 grados de libertad que cuenta con 5 motores de corriente continua independientes y las piezas mecánicas fabricadas con plástico necesarias para su correcto funcionamiento. En este proyecto se han utilizado tanto el motor que forma el codo como las piezas que unirán esta articulación con el hombro.

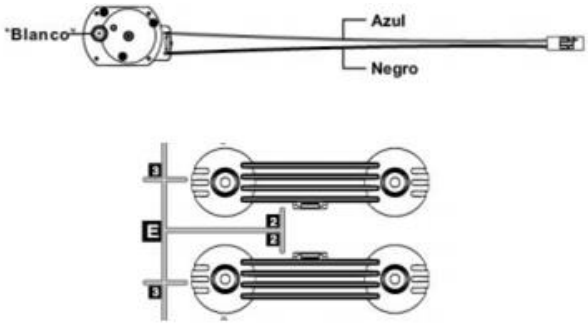


Ilustración 13 - Motor y Piezas kit brazo robot

- Piezas Lego Technic: estas piezas se han utilizado para dos partes del sistema:
    - Soporte de la tarjeta: consta de dos vigas de 8.5 cm (11 agujeros) separadas entre sí 5 cm. Para conseguir posicionarlas sobre el mismo plano horizontal se han unido con un eje de 8 cm tal y como se ve en la Ilustración 14. La distancia entre las dos vigas es la necesaria para que la tarjeta STM32F3 Discovery encaje en el sistema sin tener problemas, ni con las filas de pines ni con los jumpers que ésta tiene en su parte inferior.

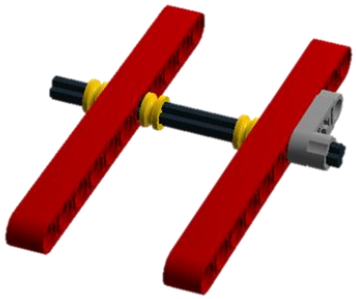


Ilustración 14 - Soporte de la tarjeta Lego Technic

    - Base delantera: esta parte tiene como objetivo mejorar la estabilidad del sistema. Consta de 4 ladrillos de 16 puntas unidos mediante una viga angular de 90º a una viga de 11.5 cm (15 agujeros). El resultado se puede observar en la Ilustración 15.

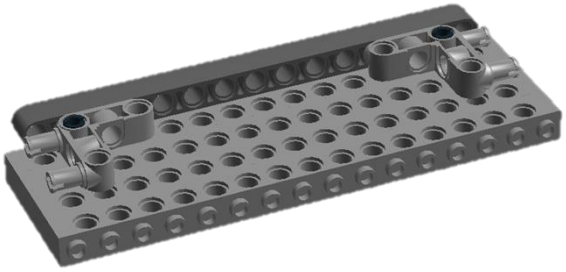


Ilustración 15 - Base delantera

- Piezas Make-Block: estas piezas tienen la característica de que las medidas de sus componentes son idénticas a las medidas de las piezas Lego Technic, por tanto, son las candidatas ideales si se pretende unir las piezas Lego con cualquier otro sistema como es este caso. En el sistema se ha utilizado una placa de 7x9 cm para unir la base delantera con el subsistema formado por el motor y el soporte para la tarjeta STM.

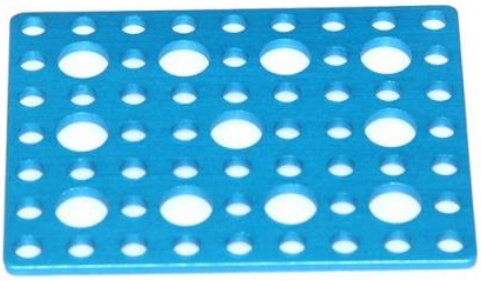


Ilustración 16 - Pieza Make-Block utilizada

- Contrapeso: con tal de evitar que el sistema se desestabilizara, durante las pruebas se decidió utilizar un contrapeso en la parte trasera. La función de contrapeso la hace una caja de cartón que contiene objetos pesados.
- Pieza de soporte de la tarjeta STM: con el fin de poder medir los ángulos con un medidor externo para poder comprobar el correcto funcionamiento del sistema y evitar soportes que pudieran dañar la placa (bridas, colas…) se decidió fabricar una pieza donde encajara el microcontrolador y a su vez dejara un espacio para apoyar dicho medidor externo y sí poder comprobar las correctas medidas. Esta pieza se fabricó con una madera de contrachapado y se tuvieron en cuenta las tiras de pines y los jumpers con los que la placa cuenta en su parte inferior El resultado final se puede ver en la Ilustración 17.

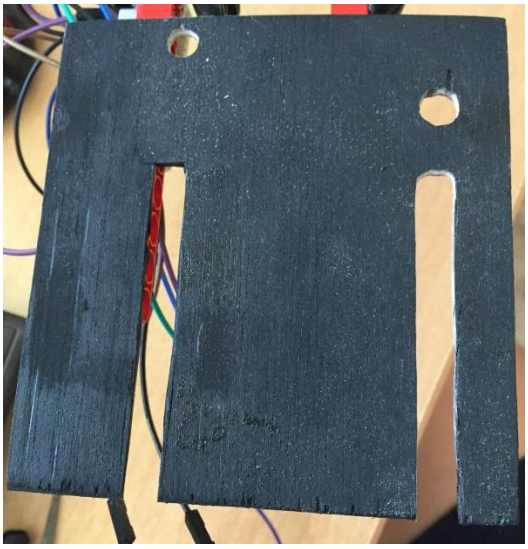


Ilustración 17 - Pieza de soporte de la tarjeta STM

4.4.2. Montaje

Como se ha comentado en el apartado anterior el prototipo del sistema de nivelación está compuesto por componentes de características muy diferentes, por tanto, a la hora de ensamblar el sistema por completo se tuvieron en cuenta ciertas características:

Para el ensamblaje de las diferentes partes del kit robótico se utilizó el material de dicho kit, ya que éste contaba con la tornillería necesaria para el montaje del brazo robot completo.

Con tal de unir esta parte con las piezas lego se utilizaron tornillos M5x30 ya que los agujeros de las piezas Lego son de este tamaño.

Como ya se ha dicho anteriormente, las piezas Make-Block permiten unir componentes de diferentes características, por ello se han utilizado tornillos M5x30 para unir esta placa con la base delantera, tornillos M4x30 para unirla con el contrapeso y la tornillería del kit de Cebek para unirlo con el motor.

Por último, para unir la pieza de soporte con las piezas Lego Technic se ha utilizado cola de contacto de la marca Ceys.

El resultado final del sistema es el mostrado en la Ilustración 18.

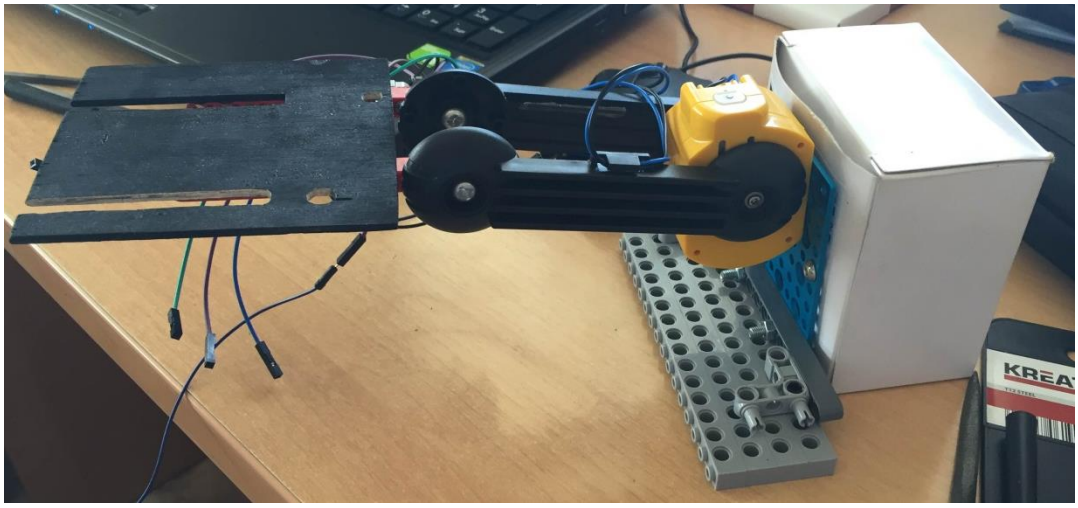


Ilustración 18 - Montaje Brazo Nivelador

4.4.3. Grados de libertad

El prototipo contará con un único grado de libertad mediante la medición del ángulo de cabeceo. La medición de este ángulo está asociada y depende enteramente de los sensores de la placa.

El rango de movimiento de la medición del ángulo es de -40º hasta 40º para la rotación sobre el eje Y.

4.5. Control de motores de Corriente Continua

El motor que utilizaremos en el prototipo del nivelador es de corriente continua. Para controlar este tipo de motores desde la placa, se usará un driver para motores (L293D) para proporcionarles la corriente necesaria para un correcto funcionamiento ya que las salidas de la tarjeta solo suministran 40 mA.

El L293D tiene dos puentes H y proporciona hasta 600 mA al motor pudiéndose alimentar por una fuente externa.

Ilustración 19 - Driver L293D

Mediante el driver podemos realizar la conexión de dos motores conectados respectivamente a los pines “Y”, y controlados mediante los pines A. Para poder controlar tanto la velocidad como el sentido de giro de estos motores hay dos formas de configurar estos pines:

- Conectar a los pines “A”, dos salidas digitales y controlar la velocidad mediante una salida PWM en el pin “Enable”.
- Conectar el pin “Enable” a una salida digital y conectar dos salidas PWM a los pines “A”.

La primera opción es interesante siempre que contemos con un microcontrolador que cuente con salidas PWM limitadas (como Arduino), sin embargo, cuenta con el problema de que si se utiliza un ciclo de trabajo muy bajo en la salida PWM, el chip puede entrar en modo de alta impedancia y bloquearse.

La segunda opción será la utilizada en este proyecto ya que el número de salidas que se pueden configurar como PWM es superior al número de salidas que necesitamos. Utilizando este método conseguimos una velocidad más baja en el motor ya que, en ningún momento, el driver entra en alta impedancia.

Además, el driver permite alimentación externa para los motores en V_{CC2}. Para esta aplicación solo controlaremos un motor y por tanto solo utilizaremos uno de los puentes H.

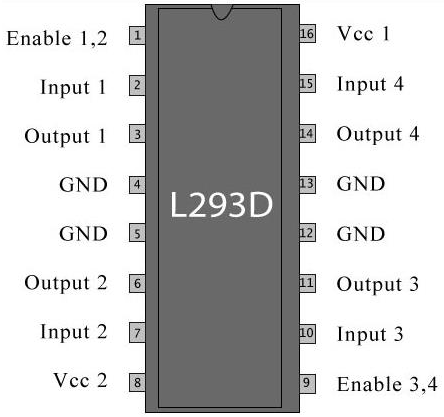


Ilustración 20 - Conexiones Driver L293D

El sentido de giro de los motores queda determinado por la combinación que aparece en los pines de entrada, así tenemos:

	1A	2 A	ESTADO
Motor 1 conectado a 1Y-2Y Enable 1 – 2 Activado	0	0	Parado
	0	1	Giro
	1	0	Giro Inverso
	1	1	Parado

	3A	4 A	ESTADO
Motor 1 conectado a 3Y-4Y Enable 1 – 2 Activado	0	0	Parado
	0	1	Giro
	1	0	Giro Inverso
	1	1	Parado

Tabla 4 - Combinaciones entradas Driver L293D y sentidos de giro

4.6. Circuito de control

El circuito de control necesario para controlar un motor de corriente continua consta de 1 driver L293D y la tarjeta STM32F3 Discovery. El circuito completo puede encontrarse en el Plano 3.

Tal y como se ve en dicho plano, se ha utilizado el puente en H correspondiente a los pines 1-7 en vez de utilizar el correspondiente a los pines 9-15. Se utilizó esta parte de pines por mayor comodidad de conexión, sin embargo, esta decisión no tiene mayor importancia ya que ambos puentes son simétricos y por tanto ambos realizan la misma función.

Por otro lado, la entrada de activación del driver (pin *Enable*) está conectada al pin PB9. Este pin está configurado como salida digital de ámbito general y se utiliza para conseguir que el driver se active únicamente cuando el programa empiece a ejecutarse ya que hasta que este pin no tome un nivel lógico alto, el puente no estará en funcionamiento.

Las entradas del driver están conectadas a los pines PD14 y PD15. Estos pines están configurados como salidas PWM ya que así se podrá controlar la velocidad a la que debe moverse el motor. Es importante poder controlar la velocidad ya que, en el prototipo construido, el recorrido del motor es limitado y una velocidad alta podría dañar algunos de los componentes electrónicos o mecánicos. Además, con tal de conseguir que la nivelación sea lo más exacta posible, es necesario que la velocidad no sea elevada ya que de no ser así, el recorrido de frenado utilizado para contrarrestar la inercia del motor es muy elevado y la nivelación no es correcta.

La alimentación externa del Driver, a 6 V, se ha realizado con la salida de este voltaje de una fuente de alimentación externa, ya que si se utilizaba la alimentación de 5 V de la propia placa esta se bloqueaba al no poder suministrar la corriente necesaria. Esta alimentación se conecta al pin V_{CC2} del driver. Para la correcta alimentación del driver las puestas a tierra del componente se han conectado al pin GND de la tarjeta.

Por último cabe mencionar que la salida V_{cc1} del driver se ha conectado al pin PB5 de la tarjeta STM32F3. Esta conexión se ha realizado para controlar la activación del componente, es decir, hasta que dicho pin no está a nivel alto el chip no está activado y por tanto no está en funcionamiento. Esta conexión se realizó ya que, durante la realización de las primeras pruebas, las corrientes de fuga provenientes del motor y de la tarjeta alimentaban de forma indeseada el driver, consiguiendo activarlo y provocando un movimiento aleatorio del sistema. Este pin está configurado como salida digital de carácter general.

4.6.1. Configuración del motor en el software

Finalmente comentaremos brevemente la forma como se ha configurado el control del motor.

Tal y como se ha explicado anteriormente el motor es controlado por dos señales PWM generadas por la placa STM32F303. Para ello se han configurado los canales 3 y 4 del Timer 4 cuya salida son los pines PD14 y PD15.

```
static void MX_TIM4_Init(void)
{

  TIM_MasterConfigTypeDef sMasterConfig;
  TIM_OC_InitTypeDef sConfigOC;

  htim4.Instance = TIM4;
  htim4.Init.Prescaler = 24;
  htim4.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim4.Init.Period = 255;
  htim4.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  if (HAL_TIM_PWM_Init(&htim4) != HAL_OK)
  {
    Error_Handler();
  }

  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim4, &sMasterConfig) != HAL_OK)
```

Vemos en el código las características del Timer que se han comentado (periodo de 255, prescaler de 24…)

```
 sConfigOC.OCMode = TIM_OCMODE_PWM1;
 sConfigOC.Pulse = 0;
 sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
 sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
 if (HAL_TIM_PWM_ConfigChannel(&htim4, &sConfigOC, TIM_CHANNEL_4) != HAL_OK)
 {
   Error_Handler();
 }

 HAL_TIM_MspPostInit(&htim4);
```

Aquí vemos como seleccionamos el canal 4 (ligado al pin PD15), con el canal 3 se ha realizado la misma operación.

El código de inicialización de los pines los encontramos en la función *HAL_TIM_MsPostInit(&htim4).* Aquí podemos ver su inicialización

```
void HAL_TIM_MspPostInit(TIM_HandleTypeDef* htim)
{

  GPIO_InitTypeDef GPIO_InitStruct;
  if(htim->Instance==TIM4)
  {
  /* USER CODE BEGIN TIM4_MspPostInit 0 */

  /* USER CODE END TIM4_MspPostInit 0 */

    /**TIM4 GPIO Configuration
    PD15     ------> TIM4_CH4
    PD14     ------> TIM4_CH3
    */
    GPIO_InitStruct.Pin = GPIO_PIN_15 | GPIO_PIN_14;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    GPIO_InitStruct.Alternate = GPIO_AF2_TIM4;
    HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);
```

Estas inicializaciones se han realizado de forma automática por el software STM32CubeMX, cuyas características se comentarán en el apartado 4.7.1.

Finalmente, para activar el PWM en la función "main" se introduce la siguiente instrucción:

```
// ACTIVAMOS  EL PWM  PD15, PD14

  HAL_TIM_PWM_Start(&htim4,TIM_CHANNEL_4);
  HAL_TIM_PWM_Start(&htim4,TIM_CHANNEL_3);
  /* USER CODE END 2 */
```

Las funciones que gobiernan el movimiento de los motores se encuentran en el fichero *motores.c.*

La principal función es *void moverMotor(int sentido,int velocidad)* que nos moverá el motor en sentido horario (sentido=1) o en sentido anti horario (sentido=2) a la velocidad pasada como parámetro. Parte de código es el siguiente:

```
void moverMotor(int sentido)
{
   int velocidad;
  velocidad= (sentido==2) ? VELOCIDAD_SUBIDA : VELOCIDAD_BAJADA;

    sentido_giro=sentido;
  // ponemos el sentido
   switch(sentido){
     case 1: // bajando
         __HAL_TIM_SetCompare(&htim4,TIM_CHANNEL_3,255);
        HAL_Delay(10);
        __HAL_TIM_SetCompare(&htim4,TIM_CHANNEL_4,0);
        HAL_Delay(10);
        __HAL_TIM_SetCompare(&htim4,TIM_CHANNEL_3,velocidad);
        HAL_Delay(10);
        __HAL_TIM_SetCompare(&htim4,TIM_CHANNEL_4,0);
        HAL_Delay(10);
          break;
     case 2: //subiendo
       __HAL_TIM_SetCompare(&htim4,TIM_CHANNEL_4,255);
        HAL_Delay(10);
        __HAL_TIM_SetCompare(&htim4,TIM_CHANNEL_3,0);
        HAL_Delay(10);
        __HAL_TIM_SetCompare(&htim4,TIM_CHANNEL_4,velocidad);
        HAL_Delay(10);
        __HAL_TIM_SetCompare(&htim4,TIM_CHANNEL_3,0);
        HAL_Delay(10);
          break;
   }
 }   // final de moverMotor
```

Vemos como controlamos la velocidad del motor mediante la función *__HAL_TIM_SetCompare* que nos permite determinar el ciclo útil del periodo, es decir, cuantos pulsos estará, a nivel lógico alto, la salida del pin asignado.

Para determinar el sentido de giro se asignan los pines de los motores adecuadamente. Las combinaciones para las entradas del Driver se pueden encontrar en la Tabla 4.

En el código observamos que durante los primeros 20 milisegundos, uno de los pines del motor se configura al máximo nivel permitido por el PWM. Esto se realiza para conseguir que el motor no tenga problemas en el arranque, que es el momento en el cual éste necesita más fuerza, ya que empieza el movimiento.

Para poder parar los motores se utiliza la función *paraMotor()* que pone a 0 la velocidad configurando el PWM con un ciclo de trabajo nulo.

```
void paraMotor()
{

        __HAL_TIM_SetCompare(&htim4,TIM_CHANNEL_3,0);
        __HAL_TIM_SetCompare(&htim4,TIM_CHANNEL_4,0);
}
```

Se han realizado otras funciones complementarias para manipular los motores como *subirMotor* y *bajarMotor* con el fin de poder realizar las pruebas con la plataforma. El código completo se puede consultar en el anexo 11.1.

Para terminar la parte de tratamiento de los motores, cabe indicar que tal como se ha explicado en el apartado anterior, tanto la alimentación del driver (V_{cc1}) como la activación del puente (Enable 1) se han conectado a dos pines digitales para poder controlar la activación del chip. Por tanto, tras la inicialización de todos los pines, se ha realizado la siguiente operación para activar el sistema:

```
//Activamos el pin de alimentación del Driver

HAL_GPIO_WritePin(GPIOB,GPIO_PIN_5,GPIO_PIN_SET);
HAL_GPIO_WritePin(GPIOB,GPIO_PIN_9,GPIO_PIN_SET);
```

4.7. Desarrollo del programa de control

4.7.1. STM32Cube MX

El software STM32Cube es un programa desarrollado por STMicroelectronics que ofrece la posibilidad de generación de código C para la configuración de periféricos y Timers de los dispositivos de esta familia, la generación de códigos listos para las herramientas de desarrollo y la importación directa de bibliotecas de software embebido STM32. Todo ello supone una reducción significativa del esfuerzo y tiempo de desarrollo.

Todos los códigos generados por esta aplicación se desarrollan con las librerías HAL (Hardware Abstraction Layer), lo cual supone una ventaja ya que si en un futuro se

encontrara la necesidad de cambiar de un microcontrolador a otro de la familia STM, esta tarea sería trivial y evitaría la necesidad de migración del código.

4.7.1.1. Utilización del STM32Cube MX

En primer lugar y tras seleccionar nuevo proyecto, seleccionamos la placa que utilizaremos y la línea adecuada.

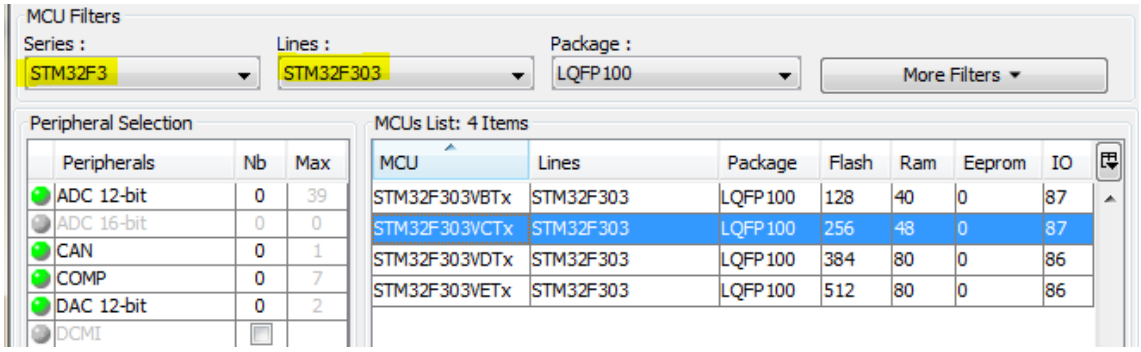


Ilustración 21 - Selección del dispositivo en el STM32 Cube

Como vemos en la Ilustración 21 se selecciona la tarjeta STM32F303VCT que es la placa que se va a utilizar en esta ocasión.

Tras hacer doble clic sobre la placa, aparece el esquema del micro. Se selecciona el Timer 4 y los canales que se van a utilizar para controlar la velocidad del motor (definidos como salidas PWM). También seleccionaremos la comunicación SPI1 para la adquisición de datos desde el giróscopo y la comunicación I_2C1 para la adquisición de datos desde el acelerómetro. Por último, seleccionaremos los pines necesarios para controlar el motor y la activación del driver.

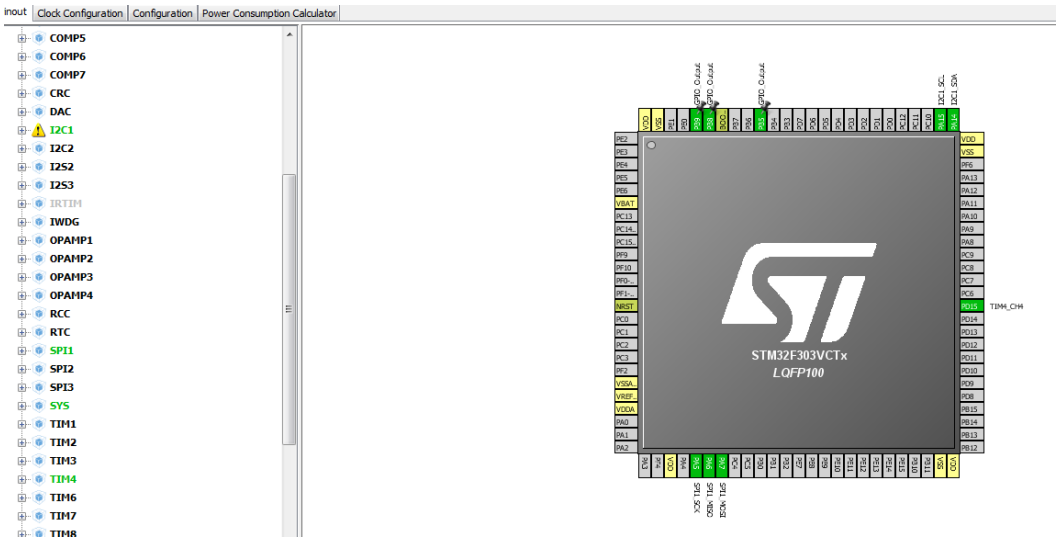


Ilustración 22 - Selección de los canales en el STM32 Cube

A continuación se configura el Timer tras entrar en el apartado de configuración:

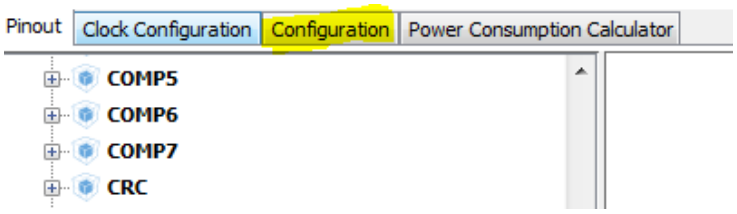


Ilustración 23 - Selección de la configuración

Seleccionamos la configuración para el Timer 4

Ilustración 24 - Selección de configuración del Timer4

Aparece la pantalla de configuración del Timer4 y del canal. Introducimos un valor de 24 en el prescaler y 255 en el apartado de periodo, (el valor de PWM podrá variar entre 0 y 255).

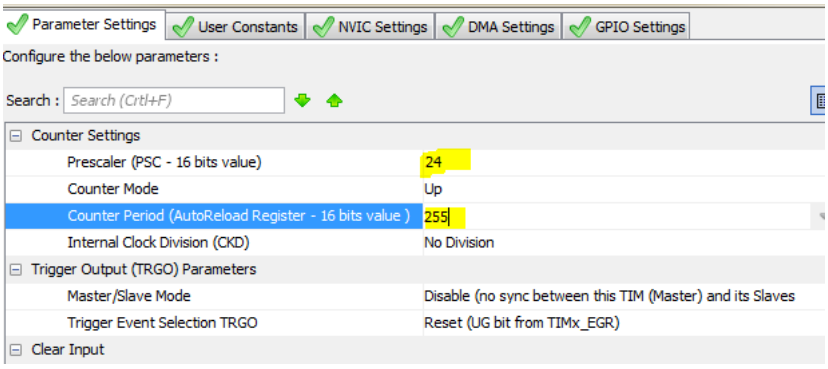


Ilustración 25 - Valores configurados para la aplicación

Tras configurar, grabamos los cambios e indicamos que el proyecto es para MDK Keil 5.

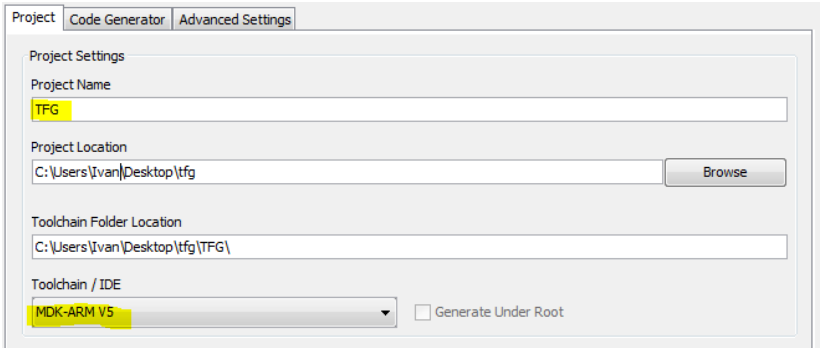


Ilustración 26 - Configuración para conseguir el código para Keil 5

En la pestaña de Code Generator, seleccionamos la opción de incluir únicamente las librerías necesarias.

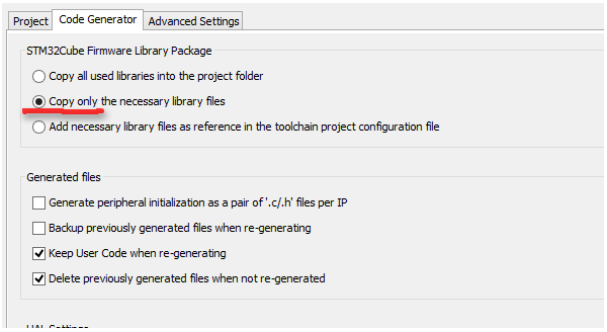


Ilustración 27 - Configuración para incluir únicamente las librerías necesarias

Tras completar la inicialización, generamos el código. Para ello pulsamos la pestaña de generación de código.

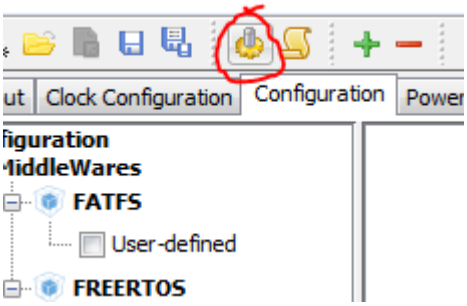


Ilustración 28 - Pestaña de generación de código

Siguiendo estos pasos el software generará un proyecto con los periféricos seleccionados inicializados. Ahora, únicamente habrá que implementar el algoritmo diseñado para el tratamiento de datos (filtro de Kalman) y para la nivelación del sistema.

Es importante revisar el código generado con el fin de verificar que la inicialización se ha realizado correctamente.

4.7.2. Configuración de pines PWM

La modulación de ancho de pulso, PWM, de una señal se trata de una técnica que logra producir el efecto de una señal analógica sobre una carga, a partir de la variación de la frecuencia y ciclo de trabajo de una señal digital. El ciclo de trabajo describe la cantidad de tiempo que la señal está en un estado lógico alto, es decir, el porcentaje del tiempo total que ésta toma para completar un ciclo completo. Se utiliza para enviar información o para modificar la cantidad de energía que se envía a una carga. Este tipo de señales son muy utilizadas en circuitos digitales que necesitan emular una señal analógica, por ejemplo, para variar la luminosidad de un led o controlar la velocidad de un motor DC.

La técnica de PWM consiste en producir un pulso rectangular con un ciclo de trabajo determinado. Este ciclo de trabajo puede variar entre 0% y 100%. Un ciclo de

trabajo del 0% significa que la señal siempre está a nivel bajo, y con un ciclo de trabajo del 100% la señal siempre se encuentra a nivel alto.

Para emular una señal analógica se cambia el ciclo de trabajo (Duty Cicle) de tal manera que el valor promedio de la señal sea el voltaje aproximado que se desea obtener, pudiendo entonces enviar voltajes entre 0 V y el máximo que soporte el dispositivo PWM (en este caso 5 V).

En la Ilustración 29 se puede observar como dependiendo del valor del ciclo de trabajo el valor promedio enviado a la carga varía.

D = Ciclo de trabajo

t = Tiempo en que la señal está en nivel lógico alto

T = Periodo

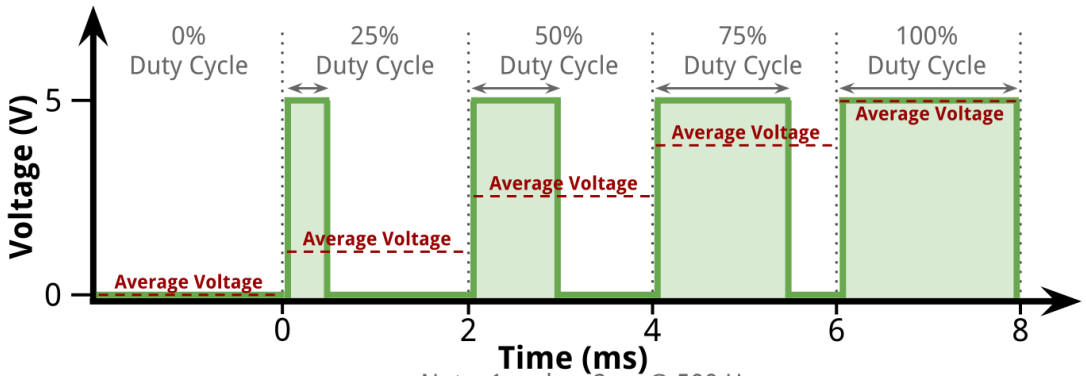


Ilustración 29 - Ejemplo PWM

La tarjeta STM32F3 Discovery dispone de Timers que permiten definir la frecuencia de trabajo. A partir del ancho de pulso indicado, que queda almacenado en un registro, se compara con un contador para activar la salida a estado alto. Algunos Timers pueden activar hasta 4 salidas PWM, una por cada canal, controlando el ciclo de trabajo de forma independiente.

No todos los Timers pueden activar salidas PWM, según el datasheet de la placa, uno de los Timers que puede generar este tipo de salidas es el Timer 4. En la Tabla 5 se muestran los pines que están conectados a cada uno de los canales de este Timer:

Timer	Canal 1	Canal 2	Canal 3	Canal 4
Timer 4	PA11, PB6, PD12	PA12, PB7, PD13	PA13, PB8, PD14	PB9, PD15, PF6

Tabla 5 - Pines conectados al Timer 4

Tal y como se ha explicado anteriormente, se ha escogido los pines PD14 y PD15 para la salida PWM.

Para configurar el Timer se ha elegido un prescaler de valor 24 y un periodo de 255, es decir, la frecuencia de trabajo estará divida por 24 y el ciclo de trabajo podrá

variar entre 0 y 255. Con estos valores se puede controlar cómodamente la velocidad del motor.

Con estos datos, el periodo de la señal PWM se calcularía de la siguiente manera:

Teniendo en cuenta que esta tarjeta trabaja a una frecuencia de 72 MHz, que en el bus utilizado se divide por 2, la frecuencia del Timer 4 es de 36 MHz, por tanto, ajustando este valor con el prescaler seleccionado:

$$f_{TIMER4} = \frac{36MHz}{24} = 1.5\ MHz \quad (13)$$

Como en cada periodo tenemos 255 divisiones, la duración de éstas será de 666.66 nanosegundos.

La frecuencia de la señal PWM se calcula como:

$$T_{PWM} = 666.66\ ns * 255 = 170\ \mu s \rightarrow f_{PMW} = 5.88\ kHz \quad (14)$$

El valor escogido de prescaler se ha tomado de forma aleatoria ya que el objetivo era poder distinguir claramente las salidas PWM sin preocuparnos del valor exacto de la frecuencia de la señal. No obstante, en otros proyectos si podría ser relevante el valor de la frecuencia PWM con lo que se debería de calcular el prescaler y el periodo de forma más precisa.

4.7.3. Programa Realizado

El desarrollo del programa es muy simple en este caso, aunque sería deseable establecer algún tipo de menú, para poder escoger entre las diferentes opciones, objetivo que queda para futuras ampliaciones del proyecto.

El diagrama de flujo del programa principal es:

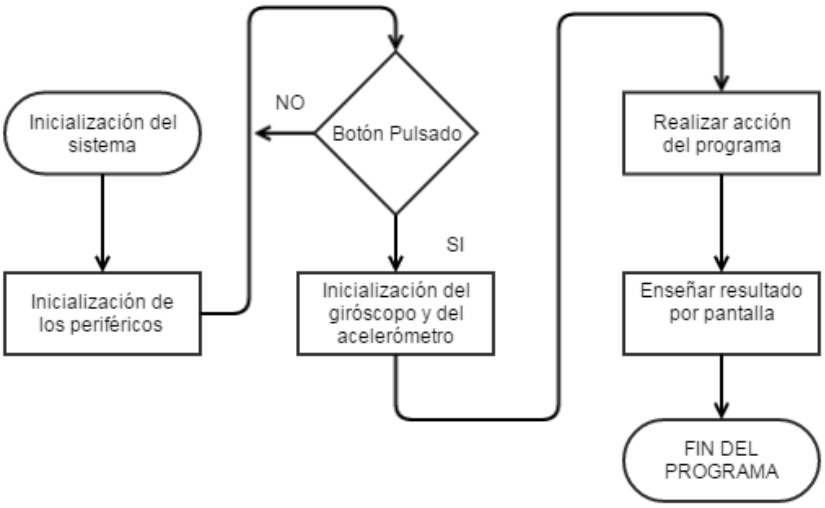


Diagrama de Bloques 2 - Programa principal

Dentro de este programa principal las diferentes opciones realizadas son:

- Calculo del ángulo mediante el acelerómetro
- Calculo del ángulo mediante el giróscopo
- Calculo del ángulo mediante la fusión de los datos del acelerómetro y del giróscopo, utilizando el filtro de Kalman.
- Nivelación de la plataforma utilizando únicamente los datos proporcionados por el acelerómetro.
- Nivelación de la plataforma utilizando únicamente los datos proporcionados por el giróscopo.
- Nivelación de la plataforma utilizando la fusión de los datos proporcionados por al acelerómetro y giróscopo mediante el filtro de Kalman.

Para el cálculo de los ángulos utilizando datos del acelerómetro o del giróscopo exclusivamente, también se ha utilizado el filtro de Kalman para calcular los datos reales. En el primer caso, suponiendo que la plataforma estaba estática y que el ángulo debía permanecer constante, se realizan 5 medidas para estimar correctamente el ángulo mediante el citado filtro. En el segundo caso se supone constante la velocidad angular y ante la dificultad de estimar la deriva del giróscopo se introduce ésta como variable de estado que será estimada en cada momento, formando el vector de estado en este caso velocidad angular, y deriva giróscopo. Todos los detalles sobre el filtro de Kalman utilizado en cada una de las opciones está desarrollado en los apartados 4.7.3.2, 4.7.3.4 y 4.7.3.6.

A continuación se explicará la implementación de cada una de las opciones con detalle.

4.7.3.1. Cálculo del ángulo de inclinación mediante el acelerómetro

En esta opción se calcula el ángulo que ha girado la plataforma alrededor del eje Y (*pitch*) medido por el acelerómetro del LSM303DLHC (características en el apartado 3.2.4.2).

Para implementar las funciones del acelerómetro se ha creado un fichero *lsm303dlhc.c* y su correspondiente *lsm303dlhc.h*.

En el fichero lsm303dlhc.h se han definido los registros propios del acelerómetro, así como las constantes que serán utilizadas por el acelerómetro. Parte de este fichero sería

```
/**********************MAPA DE REGISTROS DEL CHIP*****************/
/* Exported constant IO ------------------------------------------------------*/
#define ACC_I2C_ADDRESS                      0x32
#define MAG_I2C_ADDRESS                      0x3C

/* Acceleration Registers */
#define LSM303DLHC_WHO_AM_I_ADDR             0x0F  /* device identification register */
#define LSM303DLHC_CTRL_REG1_A               0x20  /* Control register 1 acceleration */
#define LSM303DLHC_CTRL_REG2_A               0x21  /* Control register 2 acceleration */
#define LSM303DLHC_CTRL_REG3_A               0x22  /* Control register 3 acceleration */
#define LSM303DLHC_CTRL_REG4_A               0x23  /* Control register 4 acceleration */
#define LSM303DLHC_CTRL_REG5_A               0x24  /* Control register 5 acceleration */
#define LSM303DLHC_CTRL_REG6_A               0x25  /* Control register 6 acceleration */
#define LSM303DLHC_REFERENCE_A               0x26  /* Reference register acceleration */
#define LSM303DLHC_STATUS_REG_A              0x27  /* Status register acceleration */
#define LSM303DLHC_OUT_X_L_A                 0x28  /* Output Register X acceleration */
#define LSM303DLHC_OUT_X_H_A                 0x29  /* Output Register X acceleration */
#define LSM303DLHC_OUT_Y_L_A                 0x2A  /* Output Register Y acceleration */
#define LSM303DLHC_OUT_Y_H_A                 0x2B  /* Output Register Y acceleration */
#define LSM303DLHC_OUT_Z_L_A                 0x2C  /* Output Register Z acceleration */
#define LSM303DLHC_OUT_Z_H_A                 0x2D  /* Output Register Z acceleration */
```

Pasando al fichero lsm303dlhc.c nos encontramos con las funciones propias del acelerómetro.

Las primeras funciones implementadas son las que nos permiten interactuar con el acelerómetro a través del bus I2C.

I2C_WriteData(uint16_t Addr, uint8_t Reg, uint8_t Value) Esta función escribe un byte en el registro *Reg* del dispositivo de dirección *Addr* (para el acelerómetro 0x32), utilizando el protocolo I2C mediante la función de la librería HAL *HAL_I2C_Mem_Write* . Posteriormente se deja un delay de 2 ms a fin de que la comunicación se establezca con éxito.

uint8_t I2C_ReadData(uint16_t Addr, uint8_t Reg) Función que devuelve el valor almacenado en el registro *Reg* del dispositivo cuya dirección es la pasada en *Addr*, por ejemplo, para saber el valor almacenado en el registro *0x20* del acelerómetro haríamos una llamada del tipo *value=I2C_ReadData(0x32, 0x20).* En la variable *value* tendríamos el valor almacenado en el registro *0x20* del dispositivo cuya dirección es *0x32*. Para su implementación se ha utilizado otra función de las librerías HAL *HAL_I2C_Mem_Read*.

Se han implementado otras funciones complementarias para manipular el bus I2C como *I2C_Error ()*.

A continuación se comentarán las funciones propias del acelerómetro.

HAL_StatusTypeDef InicializarAcelerometro(void) Función cuya función es la de inicializar el acelerómetro escribiendo en los registros correspondientes del acelerómetro los valores adecuados a su configuración. No acepta ningún parámetro y devuelve una variable *HAL_StatusTypeDef* indicando dos opciones: si todo ha ido bien (*HAL_OK*) o si tenemos un error al inicializar el acelerómetro (*HAL_ERROR*). La configuración elegida es: modo Normal, *ODR* a 50 Hz, se activa la lectura de los 3 ejes, (sería suficiente para este trabajo activar solo el eje Y pero a fin de facilitar la calibración se activan los 3 ejes), fondo de escala 2g, bit Endian en LSB, alta resolución etc. A continuación se configura el filtro, dejando los valores por defecto (esta configuración se puede cambiar tal como indica el datasheet.). Finalmente se escriben los valores en los registros del acelerómetro correspondientes. El diagrama de flujo que sigue esta función es el siguiente:

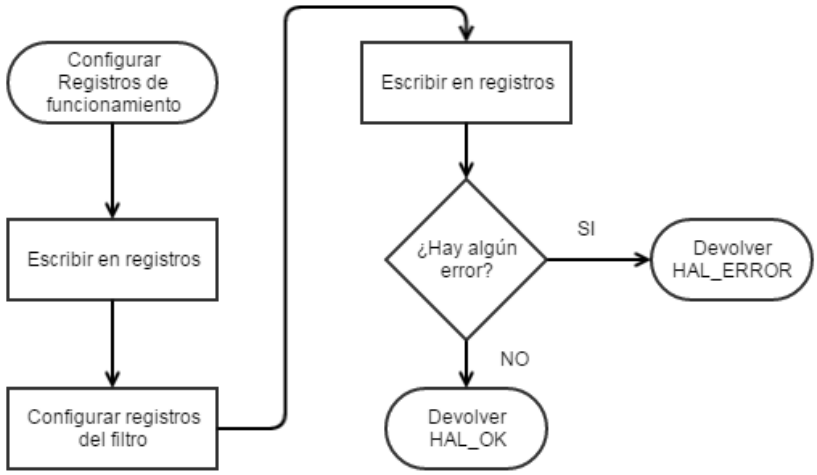


Diagrama de bloques 3 - Inicialización del acelerómetro y del giróscopo

void Leer_Aceleracion_XYZ(int16_t pData)* Esta es la función principal del acelerómetro: escribe en la variable *pData* las aceleraciones en los ejes XYZ (al pasar la dirección se guardará el valor en la variable pasada). Para nuestra aplicación nos

interesará el valor *pData[1]* que es la aceleración medida alrededor del eje Y, sin embargo a fin de calibrar el aparato leeremos los 3 ejes XYZ. Para conseguir una correcta calibración nos interesa leer el eje Z ya que sabemos que su valor valdrá 1g y en el eje X y eje Y debe ser 0g. Teniendo en cuenta las diferencias de los valores leídos con los valores que se han comentado, se calculará el error de bias del acelerómetro y así se podrá corregir los datos leídos en cada uno de los ejes.

Dentro de esta función en primer lugar leemos los registros de control a fin de conocer la configuración del dispositivo.

```
ctrlx[0] =Leer_Byte_Acelerometro(ACC_I2C_ADDRESS, LSM303DLHC_CTRL_REG4_A);
ctrlx[1] = Leer_Byte_Acelerometro(ACC_I2C_ADDRESS, LSM303DLHC_CTRL_REG5_A);
```

A continuación se leen los seis registros correspondientes a los bytes bajos y bytes altos de los tres ejes:

```
buffer[0] = Leer_Byte_Acelerometro(ACC_I2C_ADDRESS, LSM303DLHC_OUT_X_L_A);
buffer[1] = Leer_Byte_Acelerometro(ACC_I2C_ADDRESS, LSM303DLHC_OUT_X_H_A);
// eje Y
buffer[2] = Leer_Byte_Acelerometro(ACC_I2C_ADDRESS, LSM303DLHC_OUT_Y_L_A);
buffer[3] = Leer_Byte_Acelerometro(ACC_I2C_ADDRESS, LSM303DLHC_OUT_Y_H_A);
// eje Z
buffer[4] = Leer_Byte_Acelerometro(ACC_I2C_ADDRESS, LSM303DLHC_OUT_Z_L_A);
buffer[5] = Leer_Byte_Acelerometro(ACC_I2C_ADDRESS, LSM303DLHC_OUT_Z_H_A);
```

Finalmente, y según la configuración leída de los registros de control, transformaremos los valores leídos a valores de aceleración:

Primero verificamos el bit Endian, según sea MSB o LSB transformamos los 2 bytes a un entero de 16 bits en complemento a 2, el código utilizado es:

```
/* pasamos a 16 bits, (segun el bit endian del reg. 4, por defecto tenemos el */
if(!(ctrlx[0] & LSM303DLHC_BLE_MSB))
{
  // modo LSB (por defecto, El byte _L_A es el más bajo)
  for(i=0; i<3; i++)
  {
    pnRawData[i]=((int16_t)((uint16_t)buffer[2*i+1] << 8) + buffer[2*i]);
  }
}
else /* Big Endian Mode */
{
  for(i=0; i<3; i++)
  {
    pnRawData[i]=((int16_t)((uint16_t)buffer[2*i] << 8) + buffer[2*i+1]);
  }
}
```

A continuación se realiza la correspondiente corrección según el modo de almacenamiento.

```
if(ctrlx[1] & 0x40)
   cDivider=64;
else
    cDivider=16;

// en pnRawData[0]-> aceleracion eje X (medida bruta del alecelerómetro)
// en pnRawData[1]-> aceleracion eje Y (medida bruta del alecelerómetro)
// en pnRawData[2]-> aceleracion eje Z (medida bruta del alecelerómetro)
for(i=0;i<3;i++)
   pnRawData[i]=(float)pnRawData[i]/cDivider;
```

Como vemos, según sea MSB o LSB se divide el resultado por una constante a fin de ajustar los valores.

Finalmente se define la sensibilidad según el fondo de escala, multiplicando el valor resultante por dicha sensibilidad que nos transformará el valor en mg (miligravedad).

```
switch(ctrlx[0] & LSM303DLHC_FULLSCALE_16G)
{
case LSM303DLHC_FULLSCALE_2G:
  sensibilidad= LSM303DLHC_ACC_SENSITIVITY_2G;
  break;
case LSM303DLHC_FULLSCALE_4G:
  sensibilidad= LSM303DLHC_ACC_SENSITIVITY_4G;
  break;
case LSM303DLHC_FULLSCALE_8G:
  sensibilidad= LSM303DLHC_ACC_SENSITIVITY_8G;
  break;
case LSM303DLHC_FULLSCALE_16G:
  sensibilidad= LSM303DLHC_ACC_SENSITIVITY_16G;
  break;
}

/* Obtain the mg value for the three axis */
for(i=0; i<3; i++)
{
  pData[i]=(float)(pnRawData[i]*sensibilidad-Acelerometro_bias[i]) ;
}
```

Tal y como se observa en el código anterior, al valor obtenido tras todas las transformaciones se le resta el error de bias. Este valor es determinado de forma externa a esta función, exigiendo que en un suelo plano, la aceleración en los ejes X e Y debe ser nula, y en el eje Z 1000 (1000 mg es decir 1g). Para ello se ha implementado una función que realiza esta función expresamente, en la cual poniendo la placa en posición horizontal se realizan 600 medidas, se calcula la media, y se impone la condición anterior.

```
void tomarBiasAcelerometro()
{
  int i;
  int16_t pD[3]={0};
  double sum[3]={0.0};

  for(i=0;i<5;i++)              //leemos los 5 primeros y despreciamos
      Leer_Aceleracion_XYZ(pD);

  for(i=0;i<MUESTRAS_BIAS;i++){

    printf("dato bias:%d\n",i);
     Leer_Aceleracion_XYZ(pD);
    sum[0]+=pD[0];sum[1]+=pD[1];sum[2]+=pD[2];


  }
    sum[0]/=MUESTRAS_BIAS;sum[1]/=MUESTRAS_BIAS;sum[2]/=MUESTRAS_BIAS;
    sum[2]-=1000.0;   //ejez tiene que tener 1000 mg
   printf("\n dato bias ejeX:%lf\n",sum[0]);
  printf("\n dato bias ejeY:%lf\n",sum[1]);
  printf("\n dato bias ejeZ:%lf\n",sum[2]);
```

Estos valores se guardan y se introducen manualmente en la función de inicialización del acelerómetro, teniéndose en cuenta el resto de ocasiones que utilizaremos la función de lectura, eliminando de esta manera el error innato a la medida con el acelerómetro. El diagrama de flujo de la función de lectura sería:

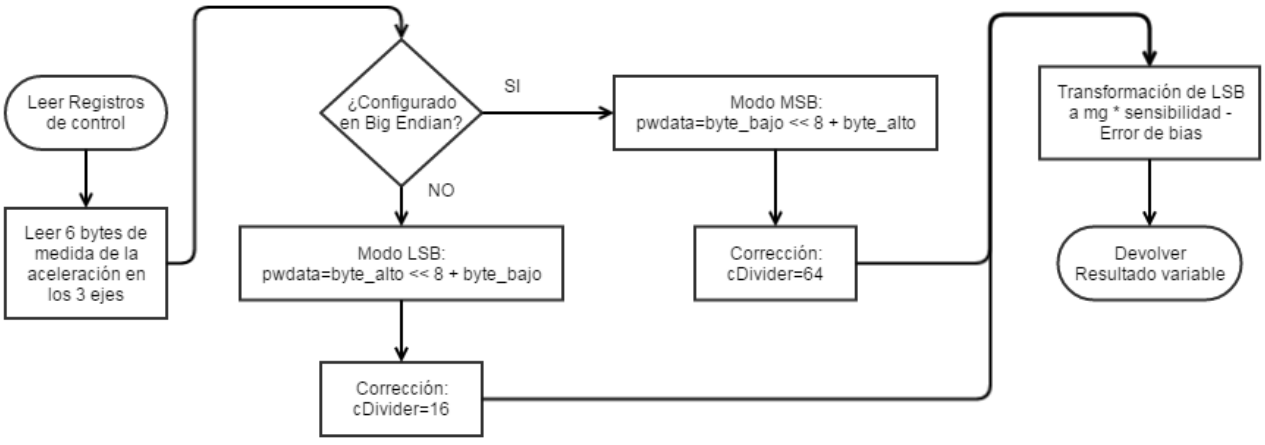


Diagrama de bloques 4 - Leer aceleración

*void calculoAngulos(double *p, double *r)* Esta función se ha incluido en el fichero *motores.c* y nos permitirá calcular los dos ángulos, el pitch y el row, a partir de las aceleraciones. Para ello, primero leemos las aceleraciones con la función comentada anteriormente y calculamos los ángulos. Esta medida se devuelve en radianes y, por tanto, para conseguir que las mediciones se realicen de forma cómoda, la transformamos a grados. Finalmente los almacenamos en las variables introducidas como parámetros (pasadas por referencia, por lo tanto mantendrán el valor que almacenemos en ellas). Por

último, se divide por mil los valores devueltos por la función de leer las aceleraciones para conseguir un valor en medida de gravedad.

```
void calculoAngulos(double *p, double *r){
int16_t pData[3];

  Leer_Aceleracion_XYZ(pData);

  *r=asin(pData[1]/1000.0);
  *p=asin(pData[0]/1000.0);

  (*p)=(*p)*180/PI;
  (*r)=(*r)*180/PI;
}
```

De esta manera obtenemos el ángulo de inclinación que forma la plataforma, utilizando exclusivamente los datos proporcionados por el acelerómetro. Estos datos son pasados por el filtro de Kalman para el acelerómetro, que nos proporcionará el valor mejor estimado de ángulo. El tratamiento del filtro de Kalman para el acelerómetro ha sido tratado en capítulos anteriores.

Comentar en este apartado que se han implementado funciones particulares para leer exclusivamente el eje Y, simplificando enormemente las funciones. La razón de su implementación ha sido la necesidad de optimizar el tiempo entre las lecturas de medidas de ángulos que se introducían en el filtro de Kalman para, así, conseguir medidas del ángulo final más fiables.

Finalmente tenemos la función *void AnguloAcelerometro(void)* que es una de las funciones que se llama desde el programa principal. Esta función es la encargada de mostrar el ángulo medido, tanto antes como después de pasar el filtro de Kalman. El diagrama de bloques que sigue esta función es el siguiente

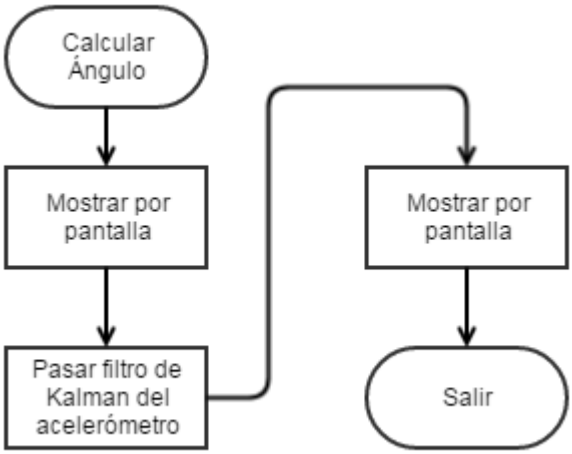


Diagrama de bloques 5 - Medir ángulo de inclinación sólo con el acelerómetro

4.7.3.2. Filtro de Kalman para el acelerómetro

En este caso se trata de estimar una magnitud que permanece constante, que es precisamente el ángulo que ha girado la plataforma alrededor del eje Y (pitch). En este primer caso, este ángulo no cambia con el tiempo por lo que debemos de estimar mediante el filtro de Kalman una magnitud que permanece constante.

Si estimamos una magnitud constante, el vector de estado tan solo estará formado por una magnitud (la que hay que estimar).

El modelo lineal que regirá el proceso de cálculo del ángulo es el siguiente:

$$angulo_k = angulo_{K-1} \quad (15)$$

Se tiene el acelerómetro como parte del sistema, comportándose como un observador de éste. De esta manera la medida entregada por el acelerómetro sirve para determinar el error suscitado entre el ángulo estimado y el medido por el acelerómetro. Teniendo esto en cuenta, se tiene el siguiente modelo para el observador:

$$z_K = Hx_k^- + v_K \quad (16)$$

Claramente se aprecia que la medida entregada por el acelerómetro presenta un cierto margen de error (v_K) con respecto al valor estimado.

$$v_K = z_K - Hx_k^- \quad (17)$$

A esta diferencia también se le denomina como *innovación*, por el papel que juega en el algoritmo de Kalman.

Con todo lo explicado se propone el siguiente modelo para incluirlo en el algoritmo del filtro de Kalman

$$angulo_K = angulo_{K-1} \quad (18)$$

$$z_K = H\widehat{x_k^-} + v_K \quad (19)$$

Donde z_K es el ángulo entregado por el acelerómetro

4.7.3.2.1. Implementación del Filtro de Kalman para el acelerómetro

Dado el algoritmo de Kalman presentado en el apartado anterior, se tiene el siguiente desarrollo del Filtro para su implementación en la placa.

- Etapa de predicción

Dado

$$\hat{x}_K = A\hat{x}_{K-1} + Bu_K \quad (20)$$

En función del modelo del sistema utilizado se tiene:

$$angulo_k^- = angulo_{k-1} \quad (21)$$

Donde

$$A = 1 \mid B = 0 \mid \hat{x}_K = aceleracion_K$$

Con lo cual la implementación en la placa estaría dada por:

$$\boldsymbol{\hat{x}_K^- = \hat{x}_{K-1}} \quad (22)$$

Recordemos que el subíndice K representa el valor de la magnitud en la interacción actual, k-1 representa el valor de la magnitud en la interacción anterior del Filtro. El superíndice "-" representa que es la magnitud estimada en la etapa de predicción, es decir, falta la etapa de corrección para obtener el valor a la salida del filtro. El circunflejo "^" indica que el valor de la magnitud es una magnitud estimada.

Dado:

$$P_K^- = AP_{K-1}A^T + Q \quad (23)$$

Donde

$$P_K \; se\; inicializa\; como\; la\; identidad \;\; y \; Q = 0.001$$

Estos valores se toman como los más indicados, según [11].

En función del sistema diseñado se tiene:

$$\boldsymbol{P_K^- = P_{K-1} + Q} \quad (24)$$

Tanto P_{k-1} como Q en este caso son escalares.

- Etapa de Corrección

Recordando que la ganancia de Kalman viene dada por:

$$K_K = P_K^- H^T (HP_K^- H^T + R)^{-1} \quad (25)$$

Donde H representa la relación entre las medidas y el estado del sistema.

En nuestro caso:

$$H = 1 \mid R = 0.07$$

El valor de R lo tomamos nuevamente de la bibliografía.

Sustituyendo valores obtenemos

$$\boldsymbol{K_K = \frac{P_k^-}{P_k^- + R}} \quad (26)$$

Siendo este valor un escalar. Con lo cual la implementación en la placa para el estado corregido estaría dada por

$$\boldsymbol{\hat{x}_K = \hat{x}_K^- + K_K(z_K - \hat{x}_K^-)} \quad (27)$$

Donde $\hat{x}_K$ representa el ángulo final estimado por el filtro (valor devuelto por el filtro)

z_K Medida leída por el acelerómetro.

$\hat{x}_K^-$ Representa el ángulo estimado en la etapa de predicción, calculada anteriormente.

K_K La llamada ganancia de Kalman.

Por otra parte, hay que recalcular la matriz de covarianza del error que, según vimos, viene dada por la expresión:

$$P_K = (I - K_K H)P_K^- \quad (28)$$

Donde

$$I = Representa\ la\ identidad.$$

Teniendo esto en cuenta y según el modelo que hemos definido obtenemos:

$$\boldsymbol{P_K = (1 - K_K)P_K^-} \quad (29)$$

P_K Matriz de covarianza actual tras el Filtro

K_K Ganancia de Kalman.

P_K^- Matriz de covarianza calculada en la etapa de predicción.

Finalmente recordemos que tenemos que guardar los datos actuales (subíndice K) como los valores de las magnitudes de la interacción anterior para la próxima interacción.

$$\boldsymbol{\hat{x}_{K-1} = \hat{x}_K} \quad (30)$$

$$\boldsymbol{P_{K-1} = \hat{P}_K} \quad (31)$$

Necesitaríamos unos valores iniciales para empezar el Filtro, para ello se toma como valor inicial del ángulo la primera medida realizada por el acelerómetro, y como valor inicial de la covariancia se toma el valor 1.

En este caso, a fin de que el filtro estime correctamente el valor, se toman cinco valores (realmente se debería hacer un estudio más exhaustivo para saber cuántas mediciones se deberían realizar a fin de que convergiera el valor del ángulo, pero esto excede el objetivo del presente trabajo).

4.7.3.3. Cálculo del ángulo de inclinación mediante el Giróscopo

Como se ha comentado anteriormente la placa cuenta con el giróscopo L3GD20 cuyas características se han comentado en el apartado 0.

Para su manejo se ha implementado un fichero, L3gd20.C, con su correspondiente fichero cabecera l3gd20.h.

El fichero cabecera, igual que en el caso del acelerómetro, se utiliza para realizar las correspondientes definiciones del giróscopo que se pueden obtener del datasheet. Parte de este fichero se muestra a continuación:

```
#define DISCOVERY_SPIx_GPIO_CLK_DISABLE()      __HAL_RCC_GPIOA_CLK_DISABLE()
#define DISCOVERY_SPIx_SCK_PIN                 GPIO_PIN_5                  /* PA.05 */
#define DISCOVERY_SPIx_MISO_PIN                GPIO_PIN_6                  /* PA.06 */
#define DISCOVERY_SPIx_MOSI_PIN                GPIO_PIN_7                  /* PA.07 */
/* Maximum Timeout values for flags waiting loops. These timeouts are not based
   on accurate values, they just guarantee that the application will not remain
   stuck if the SPI communication is corrupted.
   You may modify these timeout values depending on CPU frequency and application
   conditions (interrupts routines ...). */
#define SPI_TIMEOUT_MAX                        ((uint32_t)0x1000)


/*##################### GYRO ###########################*/

/****************************************************************************/
/************************** START REGISTER MAPPING  **************************/
/****************************************************************************/
#define L3GD20_WHO_AM_I_ADDR          0x0F  /* device identification register */
#define L3GD20_CTRL_REG1_ADDR         0x20  /* Control register 1 */
#define L3GD20_CTRL_REG2_ADDR         0x21  /* Control register 2 */
#define L3GD20_CTRL_REG3_ADDR         0x22  /* Control register 3 */
#define L3GD20_CTRL_REG4_ADDR         0x23  /* Control register 4 */
#define L3GD20_CTRL_REG5_ADDR         0x24  /* Control register 5 */
#define L3GD20_REFERENCE_REG_ADDR     0x25  /* Reference register */
```

Podemos observar las diferentes definiciones de los registros utilizados, tanto propios del giróscopo como del bus que utiliza para su comunicación que en este caso es el SPI.

Pasando al fichero l3gd20.c nos encontramos con las funciones propias del giróscopo.

Las primeras funciones implementadas son las que nos permiten interactuar con el giróscopo a través del bus SPI.

static uint8_t SPIx_WriteRead(uint8_t Byte) Función que escribe y lee un byte del registro del giróscopo siguiendo el protocolo SPI, para ello utiliza la función de las librerías HAL *HAL_SPI_TransmitReceive*, función que podemos utilizar tanto para leer como para escribir un registro del giróscopo. Como parámetro se le pasa el byte que hay que escribir en el giróscopo (puede ser una dirección o un dato) y devuelve el byte leído desde el registro del giróscopo correspondiente.

Se han implementado otras funciones complementarias para manipular el bus SPI como, *void GYRO_IO_Init(void)* donde se inicializan los pines propios del giróscopo, *void GYRO_IO_Write*, *void GYRO_IO_Read* para escribir y leer a través del bus SPI, y *void SPI_Error (void)*

Pasamos a las funciones propias del giróscopo.

HAL_StatusTypeDef InicializarGiroscopo(void) Función cuya misión es la de inicializar el giróscopo escribiendo en los registros correspondientes, los valores adecuados a su configuración. No acepta ningún parámetro y devuelve una variable *HAL_StatusTypeDef* indicando si todo ha ido bien (HAL_OK) o si tenemos un error al inicializar el giróscopo (HAL_ERROR). La configuración elegida es: modo Normal, ancho de banda a 50 Hz, se activa la lectura de los 3 ejes, (sería suficiente para este trabajo activar solo el eje X, pero a fin de facilitar la calibración se activan los 3 ejes), fondo de escala 500 dps, bit Endian en LSB, alta resolución etc. A continuación se configura el filtro. En este caso el filtro no se activa por defecto (esta configuración se puede cambiar tal como indica el datasheet). Finalmente se escriben los valores en los registros del giróscopo correspondientes. El diagrama de flujo que sigue esta función sería el incluido en el Diagrama de bloques 3 utilizado para el acelerómetro, pero con los valores propios del giróscopo.

Se implementan otras funciones de cambio de configuración del giróscopo que en este proyecto no vamos a utilizar, pero dejamos preparado para futuras ampliaciones.

*void Leer_VelocidadAngular_XYZ(float *pfData)* Es la función principal del giróscopo. Escribe en la variable *pData* las velocidades angulares alrededor de los ejes XYZ (al pasar la dirección se guardará el valor en la variable pasada), nos interesará el valor *pData[0]* que es la velocidad angular alrededor del eje X..

En primer lugar leemos los registros de control con el fin de conocer la configuración del dispositivo.

```
GYRO_IO_Read(&tmpreg,L3GD20_CTRL_REG4_ADDR,1);
```

El resultado se almacena en la variable *tmpreg* que se utilizará posteriormente para poder procesar los datos leídos.

A continuación se leen los seis registros correspondientes a los bytes bajos y bytes altos de los tres ejes:

```
GYRO_IO_Read(tmpbuffer,L3GD20_OUT_X_L_ADDR,6);
```

En este caso leemos los seis registros a la vez, los bytes altos y los bytes bajos de las velocidades angulares alrededor de los ejes X, Y, Z.

Finalmente, y según la configuración leída de los registros de control, transformaremos de los valores leídos a valores de velocidad angular:

Primero verificamos el bit Endian, según sea MSB o LSB transformamos los 2 bytes a un entero de 16 bits en complemento a 2, el código utilizado es:

```
/* check in the control register 4 the data alignment (Big Endian or Little Endi
if(!(tmpreg & L3GD20_BLE_MSB))
{
  for(i=0; i<3; i++)
  {
    RawData[i]=(int16_t)(((uint16_t)tmpbuffer[2*i+1] << 8) + tmpbuffer[2*i]);
  }
}
else
{
  for(i=0; i<3; i++)
  {
    RawData[i]=(int16_t)(((uint16_t)tmpbuffer[2*i] << 8) + tmpbuffer[2*i+1]);
  }
}
```

Finalmente se define la sensibilidad según el fondo de escala, multiplicando el valor resultante por dicha sensibilidad que nos transformara el valor en mdps (miligrados/segundo).

```
switch(tmpreg & L3GD20_FULLSCALE_SELECTION)
{
case L3GD20_FULLSCALE_250:
  sensitivity=L3GD20_SENSITIVITY_250DPS;
  break;

case L3GD20_FULLSCALE_500:
  sensitivity=L3GD20_SENSITIVITY_500DPS;
  break;

case L3GD20_FULLSCALE_2000:
  sensitivity=L3GD20_SENSITIVITY_2000DPS;
  break;
}
/* Divide by sensitivity */
for(i=0; i<3; i++)
{
 // pfData[i]=(float)(((-1)*RawData[i] * sensitivity/1000)-Giroscopo_bias[i]);
  pfData[i]=(float)((-1)*RawData[i] * sensitivity/1000);
}
```

Vemos como multiplicamos por (-1) el valor leído con la finalidad de que coincida el signo de la velocidad angular medida por el giróscopo con el signo del ángulo medido por el acelerómetro.

En este caso no restamos ningún error de deriva del giróscopo, aunque tal y como se ha comentado en el apartado 3.3.2, el error de deriva del giróscopo para el cálculo de ángulos es muy elevado. La razón es porque la deriva del giróscopo suponemos que depende de muchas variables, y a fin de obtener un valor más fiable, introducimos esa magnitud junto con la velocidad angular, como variables de estado en la implementación del filtro de Kalman. Por tanto, la velocidad angular devuelta por el filtro sería la velocidad angular estimada por el filtro menos la deriva estimada por el mismo filtro.

void calculoAnguloGiroscopo(double angulo_anterior) Esta función nos devuelve, dado un ángulo inicial, el ángulo final transcurrido un cierto tiempo. Para ello primero leemos la velocidad inicial y a continuación, sabiendo el ángulo anterior, la velocidad angular alrededor del eje X y el tiempo transcurrido desde que se midió el ángulo anterior. Así podemos calcular fácilmente el ángulo final:

```
angulo=angulo_anterior+pfData[0]*(t2-tiempo1)/1000;  //
tiempo1=t2;
return angulo;
```

Para calcular el tiempo transcurrido, definimos una variable global *tiempo1* que se inicializa en el momento en que calculamos el ángulo inicial, y la variable *t2* que lo calculamos en el momento justo antes del cálculo del ángulo, empezando a contar el tiempo en el momento del cálculo del ángulo. La función completa podemos verla a continuación:

```
double CalculoAnguloGiroscopo(double angulo_anterior){
  long int t2;
  double angulo;
  float pfData[3]={0.0};

  //t1=HAL_GetTick();
  Leer_VelocidadAngular_XYZ(pfData); // leemos la velocidad angular alrededor del eje X en °/s
  // deberiamos pasar el filtro de Kalman aquí para el giróscopo.


    pfData[0]=filtroKalmanGiroscopo((double)pfData[0]);

  t2=HAL_GetTick();

//  printf("velocidad angular wx=%f wy=%f wz=%f, t1=%ld, t2=%ld,t2-t1=%ld",pfData[0],pfData[1],
  angulo=angulo_anterior+pfData[0]*(t2-tiempo1)/1000;  //phi_anterior+w_x*dt  ,w en grados/s, d
  tiempo1=t2;
  return angulo;
}
```

Vemos como tras leer las velocidades angulares, se pasa por el filtro de Kalman cuyas características ya han sido comentadas anteriormente.

Por lo tanto, para poder calcular un ángulo a través del giróscopo necesitamos conocer el ángulo inicial y la velocidad angular y, tras un tiempo, podremos calcular el ángulo final.

void AnguloGiroscopo(void) Es una de las funciones principales. Será llamada desde el programa principal y aparece en el fichero *medidas.c*. El diagrama seguido, es el mostrado en la figura siguiente.

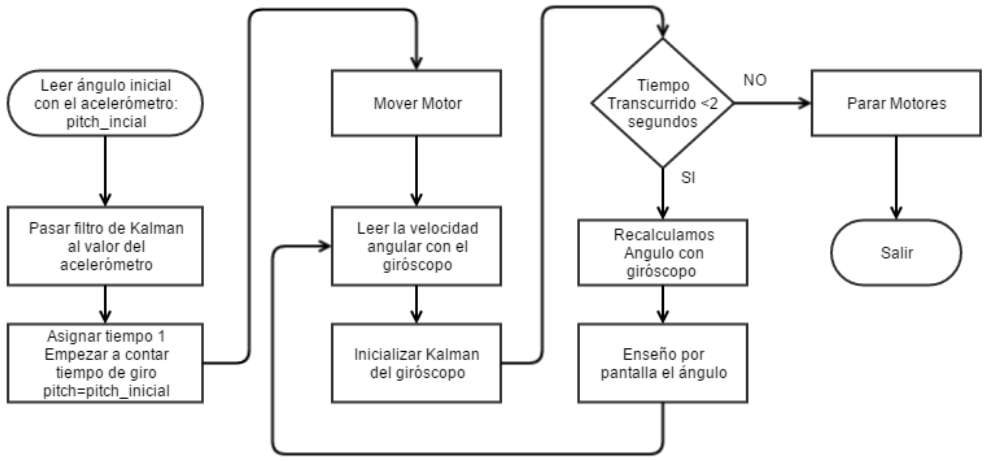


Diagrama de bloques 6 - Medir ángulo de inclinación sólo con el giróscopo

En primer lugar, medimos el ángulo inicial mediante el acelerómetro (para tener un ángulo inicial de referencia), al cual le pasamos el filtro de Kalman propio. Los motores están parados por lo que es de esperar que la medida sea fiable. A continuación se inicializa el tiempo, y para ello se utiliza la función de librería *HAL_GetTick()*, que nos devuelve el tiempo desde el inicio del programa en milisegundos. Este sería el tiempo inicial (será necesario para establecer el tiempo transcurrido). Empezamos a mover el motor (definimos un sentido de giro) y a continuación leemos la velocidad angular mediante el giróscopo, y con esa velocidad inicial, inicializamos el filtro de Kalman para el giróscopo. Inicializado todo el proceso entramos en un bucle que nos indicará el ángulo de la plataforma durante 2 segundos. Transcurridos los dos segundos se paran los motores y podemos ver por pantalla el último ángulo medido, con el cual verificaremos la exactitud del sistema.

El tiempo transcurrido lo medimos en milisegundos ya que es la medida que utiliza por defecto la función descrita anteriormente. De querer mayor exactitud deberíamos modificar la variable *SysTick()* o implementar una función específica mediante otro Timer.

Para el cálculo del ángulo se utiliza otra función llamada *double CalculoAnguloGiroscopo(double angulo_anterior)* situada en el fichero *l3gd20.c* , cuyo diagrama se muestra a continuación

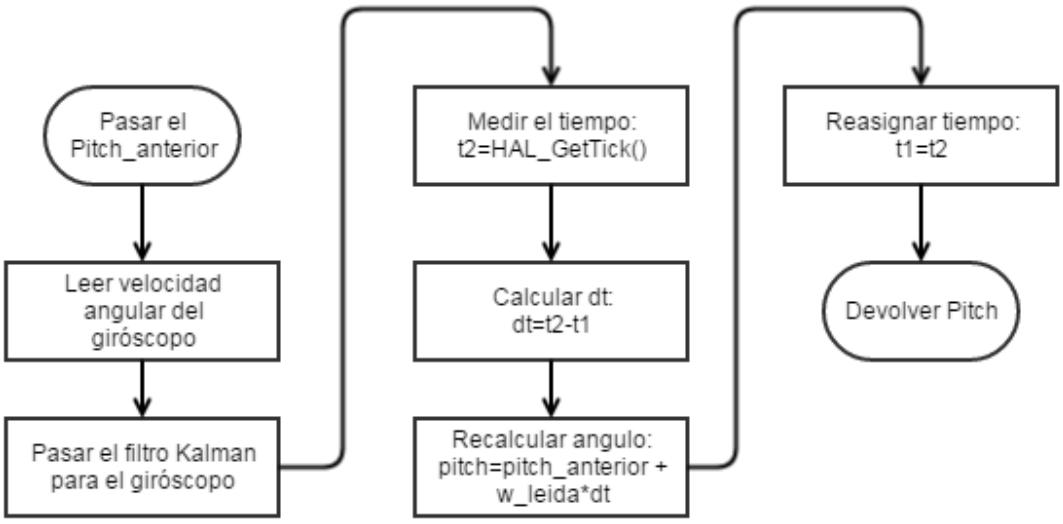


Diagrama de bloques 7- Cálculo del ángulo de inclinación con el giróscopo

A esta función se le pasa el ángulo anterior, y, calculando la velocidad angular mediante el giróscopo y estimándola mediante el filtro Kalman, y midiendo el tiempo transcurrido desde la última medida (para ello calculamos *t2*, tiempo transcurrido hasta ese momento y le restamos *tiempo1* tiempo marcado en la última medida), podemos calcular el ángulo final:

$$angulo_{final} = angulo_{incial} + velocidad_{angular} * tiempo_{transcurrido}$$

Esta función devuelve el ángulo a la función principal *Angulo_giróscopo( )*, en la cual se enseña y se controla el tiempo de movimiento de los motores.

El código completo se puede consultar en el Anexo 11.1

4.7.3.4. Filtro de Kalman para el giróscopo

En este caso también se trata de estimar una magnitud que permanece constante. Para este caso se trata de la velocidad angular con la que gira la plataforma alrededor del eje X. Supondremos que la velocidad angular no cambia con el tiempo por lo que debemos de estimar mediante el filtro de Kalman una magnitud que permanezca constante.

Por lo tanto se podría proceder como en el caso del acelerómetro, donde el filtro se reduce a magnitudes escalares. En este caso, la dificultad en estimar el error de deriva del giróscopo medido en º/s al estar en continua variación hace difícil proporcionar un valor que permanezca constante para esta variable a lo largo del tiempo. Teniendo esto en cuenta, hemos optado por tomar la variable deriva del giróscopo como una variable de estado, formando parte del vector de estados que estará bajo el algoritmo de Kalman, donde al estar sujeta a cambios provocados por todas las variables que se incluyen en el sistema, su valor de carácter aleatorio en el proceso se verá reflejado con mayor veracidad. Como modelo de esta nueva variable de estado, se tomará constante a lo largo del tiempo.

Así, el modelo del sistema quedaría definido por estas dos ecuaciones:

$$u_k = u_{K-1} - deriva_{K-1} \quad (32)$$

$$deriva_K = deriva_{K-1} \quad (33)$$

Se tiene el giróscopo como parte del sistema, comportándose como un observador de éste. De esta manera, la medida entregada por el giróscopo sirve para determinar el error suscitado entre la velocidad angular estimada y la medida por el giróscopo (como se ha visto en el capítulo sobre el filtro, la medida está dada con

cierto grado de error). Teniendo esto en cuenta se tiene el siguiente modelo para el observador

$$z_K = Hx_k^- + v_K \quad (34)$$

Claramente se aprecia que la medida entregada por el giróscopo presenta un cierto margen de error (v_K) con respecto al valor estimado

$$v_K = z_K - Hx_k^- \quad (35)$$

A esta diferencia también se le denomina como *innovación*, por el papel que juega en el algoritmo de Kalman.

Con todo lo que hemos dicho se propone el siguiente modelo para incluirlo en el algoritmo del filtro de Kalman

$$\begin{pmatrix} u^- \\ deriva^- \end{pmatrix}_K = \begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} u \\ deriva \end{pmatrix}_{K-1} \quad (36)$$

$$z_K = Hx_k^- + v_K \quad (37)$$

También resolviendo la matriz se obtiene:

$$u_k^- = u_{K-1} - deriva_{K-1} \quad (38)$$

$$deriva_K^- = deriva_{K-1} \quad (39)$$

$$z_K = Hu_k^- + v_K \quad (40)$$

Dónde:

z_k = Velocidad angular proporcionada cada medida por el giróscopo

u_k = Velocidad angular estimada

4.7.3.4.1. Implementación del filtro de Kalman del giróscopo

Dado el algoritmo de Kalman presentado en el apartado anterior, se tiene el siguiente desarrollo del Filtro para su implementación en la placa.

- <u>Etapa de Predicción</u>

Tenemos en este caso: $\hat{x}_K^- = A\hat{x}_{K-1}$

Y según el modelo anterior:

$$\begin{pmatrix} u^- \\ deriva^- \end{pmatrix}_K = \begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} u \\ deriva \end{pmatrix}_{K-1} \quad (41)$$

$$donde\ A = \begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix}$$

$$\hat{x}_K = \begin{pmatrix} vel.angular\ (u) \\ deriva \end{pmatrix}_K \quad (42)$$

Con lo cual la implementación en la placa estaría dada por:

$$x[0]_K^- = \hat{x}[0]_{K-1} - \hat{x}[1]_{K-1} \quad (43)$$

$$\hat{x}[1]_K^- = \hat{x}[1]_{K-1} \quad (44)$$

Recordemos que el subíndice *k* representa el valor de la magnitud en la interacción actual, *k-1* representa el valor de la magnitud en la interacción anterior del Filtro. El superíndice "-" representa que es la magnitud estimada en la etapa de predicción, es decir que falta la etapa de corrección para obtener el valor a la salida del filtro. El circunflejo "^" indica que el valor de la magnitud es una magnitud estimada.

Vemos como en este caso estamos tratando con vectores de dos componentes, (componentes del vector de estados del Filtro)

Dado:

$$P_K^- = A\,P_{K-1}\,A^+ + Q$$

Con $Q = \begin{pmatrix} 0.001 & 0 \\ 0 & 0.003 \end{pmatrix}$ el Q[0][0] está relacionado con el error en el modelo de la velocidad angular y el Q[1][1] está relacionado con el modelo que hemos supuesto para la deriva del giróscopo. Los valores se han tomado según [11].

En función del sistema se tiene:

$$P_K^- = \begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} P_{K-1}(0,0) & P_{K-1}(0,1) \\ P_{K-1}(1,0) & P_{K-1}(1,1) \end{pmatrix}\begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix} + \begin{pmatrix} Q(0,0) & Q(0,1) \\ Q(1,0) & Q(1,1) \end{pmatrix} =$$

$$\begin{pmatrix} P_{K-1}(0,0) - P_{K-1}(1,0) & P_{K-1}(0,1) - P_{K-1}(1,1) \\ P_{K-1}(1,0) & P_{K-1}(1,1) \end{pmatrix}\begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix} + \begin{pmatrix} Q(0,0) & Q(0,1) \\ Q(1,0) & Q(1,1) \end{pmatrix}$$

$$= \begin{pmatrix} P_{K-1}(0,0) - P_{K-1}(1,0) - P_{K-1}(0,1) + P_{K-1}(1,1) & P_{K-1}(0,1) - P_{K-1} \\ P_{K-1}(1,0) - (P_{K-1}(1,1) & P_{K-1}(1,1) \end{pmatrix} \quad ((4\ 5) + Q$$

Por tanto:

$$P_K^-(0,0) = P_{K-1}(0,0) - P_{K-1}(1,0) - P_{K-1}(0,1) + P_{K-1}(1,1) + Q(0,0) \quad (46)$$

$$P_K^-(0,1) = P_{K-1}(0,1) - P_{K-1}(1,1) + Q(0,1) \quad (47)$$

$$P_K^-(1,0) = P_{K-1}(1,0) - P_{K-1}(1,1) + Q(1,0) \quad (48)$$

$$P_K^-(1,1) = P_{K-1}(1,1) + Q(1,1) \quad (49)$$

- Etapa de Corrección

Recordando que la ganancia de Kalman viene dado por:

$$K_K = P_K^- H^T (H\, P_K^-\, H^T + R)^{-1} \quad con \quad H = (1 \quad 0), y \;\; R = 0.05$$

Donde el valor de R es tomado de la bibliografía consultada.

En este caso P_K (matriz de covarianza del error) será una matriz de 2x2, y H será un vector de 1x2 con valor (1 0).

Desarrollando la ecuación anterior queda:

$$\begin{pmatrix} K[0] \\ K[1] \end{pmatrix} = \frac{\begin{pmatrix} P_K^-(0,0) & P_K^-(0,1) \\ P_K^-(1,0) & P_K^-(0,1) \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix}}{\left((1 \quad 0) \begin{pmatrix} P_K^-(0,0) & P_K^-(0,1) \\ P_K^-(1,0) & P_K^-(1,1) \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} + R \right)} = \frac{\begin{pmatrix} P_K^-(0,0) \\ P_K^-(1,0) \end{pmatrix}}{P_K^-(0,0) + R} \quad (50)$$

$$\boxed{K(0) = \frac{\left(P_K^-(0,0)\right)}{P_K^-(0,0) + R} \quad | \quad K(1) = \frac{P_K^-(1,0)}{P_K^-(0,0) + R}} \quad (51)$$

Con lo cual la implementación en la placa para el estado corregido estaría dada por:

$$\hat{x}_K = \hat{x}_K^- + K_K(Z_K - H\hat{x}_K^-) \quad (52)$$

$$\begin{pmatrix} x_K(0) \\ x_k(1) \end{pmatrix} = \begin{pmatrix} x_K^-(0) \\ x_K^-(1) \end{pmatrix} + \begin{pmatrix} K(0) \\ K(1) \end{pmatrix} \left[Z_K - (1 \quad 0) \begin{pmatrix} x_K^-(0) \\ x_K^-(1) \end{pmatrix} \right] \quad (53)$$

$$\boxed{\boldsymbol{x_K(0) = x_K^-(0) + K(0)[Z_K - x_K^-(0)]}} \quad (54)$$

$$\boxed{\boldsymbol{x_K(1) = x_K^-(1) + K(1)[Z_K - x_K^-(0)]}} \quad (55)$$

Donde $\hat{x}_K$ representa el vector de estado final estimado por el filtro (el valor que el filtro estima como más exacto- es el valor devuelto por el filtro-) Vector de 2x1

z_K Medida leída por el giróscopo. Escalar

$\hat{x}_K^-$ representa el vector de estado estimado en la etapa de predicción, calculada anteriormente. Vector de 2x1

K_K La llamada ganancia de Kalman. Es un vector de 2x1

Por otra parte, hay que recalcular la matriz de covarianza del error, que según vimos viene dada por la expresión:

$$P_K = (I - K_K H)P_K^-$$

Donde

$$I = Representa\ la\ matriz\ \ identidad\ de\ 2x2.$$

Teniendo esto en cuenta y según el modelo que hemos definido obtenemos:

$$\begin{pmatrix} P_K(0,0) & P_K(0,1) \\ P_K(1,0) & P_K(1,1) \end{pmatrix} = \left[\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} - \begin{pmatrix} K(0) \\ K(1) \end{pmatrix}(1,0)\right]\begin{pmatrix} P_K^-(0,0) & P_K^-(0,1) \\ P_K^-(1,0) & P_K^-(1,1) \end{pmatrix} =$$

$$= \left[\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} - \begin{pmatrix} K(0) & 0 \\ K(1) & 0 \end{pmatrix}\right]\begin{pmatrix} P_K^-(0,0) & P_K^-(0,1) \\ P_K^-(1,0) & P_K^-(1,1) \end{pmatrix} =$$

$$= \left[\begin{pmatrix} 1 - K(0) & 0 \\ -K(1) & 1 \end{pmatrix}\right]\begin{pmatrix} P_K^-(0,0) & P_K^-(0,1) \\ P_K^-(1,0) & P_K^-(1,1) \end{pmatrix} \quad (56)$$

$$\boxed{\boldsymbol{P_K(0,0) = (1 - K(0)) * P_K^-(0,0)}} \quad (57)$$

$$\boxed{\boldsymbol{P_K(0,1) = (1 - K(0)) * P_K^-(0,1)}} \quad (58)$$

$$\boxed{\boldsymbol{P_K(1,0) = -K(1) * P_K^-(0,0) + P_K^-(1,0)}} \quad (59)$$

$$\boxed{\boldsymbol{P_K(1,1) = -K(1) * P_K^-(0,1) + P_K^-(1,1)}} \quad (60)$$

Dónde:

P_K matriz de covarianza actual tras el Filtro

K_k ganancia de Kalman.

P_K^- matriz de covarianza calculada en la etapa de predicción.

Finalmente recordemos que tenemos que guardar los datos actuales (subíndice *k*) como los valores de las magnitudes de la interacción anterior para la próxima interacción, es decir hay que copiar los valores actuales como los anteriores antes de salir de la función.

$$\widehat{\boldsymbol{x}}_{\boldsymbol{K-1}} = \widehat{\boldsymbol{x}}_{\boldsymbol{K}} \quad (61)$$

$$\boldsymbol{P}_{\boldsymbol{K-1}} = \widehat{\boldsymbol{P}}_{\boldsymbol{K}} \quad (62)$$

Necesitaríamos unos valores iniciales para empezar el Filtro, para ello se toma como valor inicial la velocidad angular medida de forma externa al programa (se toman 10 medidas y se toma el valor medio), según sea el movimiento de subida o de bajada toma un valor diferente. Como valor inicial de la covariancia se toma la matriz identidad I de 2x2).

4.7.3.5. Cálculo del ángulo de inclinación mediante la fusión de los datos del acelerómetro y del giróscopo

En este caso se utilizarán las funciones desarrolladas para la medida de los ángulos utilizando acelerómetro y giróscopo. Para fusionarlos se implementará un filtro de Kalman a fin de adaptarlo a las dos medidas. Se considerará la medida del giróscopo como una entrada al sistema, y el dato del acelerómetro como una medida introducida en la etapa de corrección del filtro. Como variables tendremos el ángulo medido y la deriva del giróscopo. Todo este razonamiento se expondrá con más detalle en el apartado destinado al filtro de Kalman para acelerómetro y giróscopo.

La función principal que se utiliza en este apartado es *void Angulo_Giroscopo_Acelerometro(void)* que nos irá mostrando el ángulo mientras la plataforma se mueve durante 2 segundos, nos interesará el ángulo final, que es que se podrá verificar.

El diagrama seguido para su implementación es el mostrado a continuación

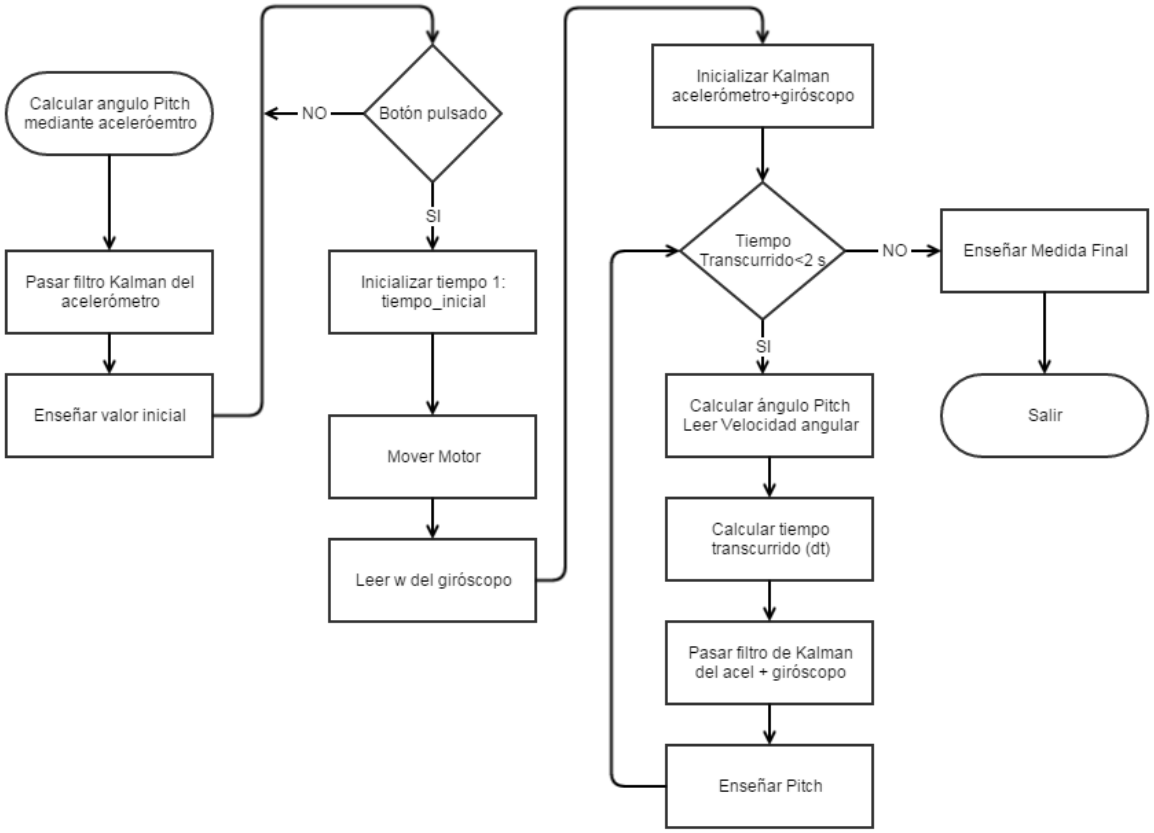


Diagrama de bloques 8 - Medir el ángulo de inclinación con acelerómetro y giróscopo

En primer lugar se calcula el ángulo inicial mediante el acelerómetro, enseñando por pantalla dicho valor inicial para su verificación. Mientras no se pulse el botón de usuario el sistema no realiza ningún cambio con el fin de que se pueda realizar esa verificación inicial. Cuando se pulsa el botón de usuario se empieza inicializando el tiempo y se comienza a mover el motor. Mediante el giróscopo se mide la velocidad angular y se inicializa el filtro de Kalman para el acelerómetro más giróscopo.

A continuación, mientras el tiempo transcurrido desde la inicialización sea menor de 2 segundos, se calculará el ángulo mediante el acelerómetro que será considerado como medida de referencia para el filtro de Kalman. Se mide la velocidad angular por medio del giróscopo que será considerada como entrada al sistema en el filtro de Kalman. Se calcula el tiempo transcurrido desde la anterior medida y se pasa tanto la medida de la aceleración, la velocidad angular como entrada y el tiempo transcurrido al filtro donde se realiza la fusión de los datos con el modelo lineal como se verá en la implementación del filtro. Una vez realizados los cálculos en el filtro, éste nos devuelve el ángulo estimado que se enseña por pantalla. Transcurridos los dos segundos se paran los motores y se sale

de la función. Vemos como la fusión de los datos aportados por el acelerómetro y del giróscopo se realiza en el modelo lineal implementado en el filtro de Kalman.

4.7.3.6. Filtro de Kalman para el giróscopo y el acelerómetro

En este caso también se trata de estimar una magnitud que no permanece constante pero que sigue un modelo lineal, el ángulo que ha girado la plataforma alrededor del eje Y (pitch). En este caso la velocidad angular supondremos que no cambia con el tiempo por lo que debemos de estimar mediante el filtro de Kalman una magnitud que sigue un modelo lineal que deberemos definir previamente.

En este caso la dificultad en estimar el error de deriva del giróscopo medido en º/s al estar en continua variación hace difícil proporcionar un valor que permanezca constante, para esta variable, a lo largo del tiempo. Teniendo esto en cuenta, hemos optado por tomar esta variable (deriva del giróscopo) como una variable más de estado formando parte del vector de estados que estará bajo el algoritmo de Kalman, donde al estar sujeta a cambios provocados por todas las variables que se incluyen en el sistema, su valor de carácter aleatorio en el proceso se verá reflejado con mayor veracidad.

En primer lugar definimos el modelo que sigue tanto el ángulo (Pitch) como el modelo que sigue la deriva del giróscopo, que son los componentes el vector de estados del Filtro. El modelo del ángulo será un momento lineal, sin embargo, el modelo seguido por la deriva del giróscopo supondremos que es una magnitud que permanece constante. Según esto tenemos:

Modelo que seguirá el ángulo

$$angulo_K^- = angulo_{K-1} - (u_K - \omega_{K-1}) \cdot dt \quad (63)$$

$$angulo_K^- = angulo_{K-1} + u_K \cdot dt - \omega_{K-1} \cdot dt \quad (64)$$

Modelo que seguirá la deriva del giróscopo

$$\omega_K^- = \omega_{K-1} \quad (65)$$

Donde en resumen tenemos:

u_K → Señal del giróscopo

dt → Tiempo de muestreo

ω → Deriva

$angulo_{K-1}$ → Ángulo anterior

$angulo_K^-$ → Ángulo estimado

ω_{K-1} Muy difícil de estimar → Pasamos variable de estado.

Tenemos:

$$\boxed{\boldsymbol{angulo_K^- = angulo_{K-1} + u_K \cdot dt - \omega_{K-1} \cdot dt} \quad | \quad \boldsymbol{\omega_K^- = \omega_{K-1}}} \qquad (66)$$

Se tiene el acelerómetro como parte del sistema comportándose como un observador de éste y el giróscopo que nos proporciona una entrada al sistema. De esta manera la medida entregada por el acelerómetro sirve para determinar el error suscitado entre el ángulo estimado y el medido por el acelerómetro. Teniendo esto en cuenta se tiene el siguiente modelo para el observador

$$z_K = Hx_k^- + v_K \qquad (67)$$

Claramente se aprecia que la medida entregada por el acelerómetro presenta un cierto margen de error (v_k) con respecto al valor estimado

$$v_K = z_K - Hx_k^- \qquad (68)$$

A esta diferencia también se le denomina como *innovación*, por el papel que juega en el algoritmo de Kalman.

Con todo lo que hemos dicho se propone el siguiente modelo para incluirlo en el algoritmo del filtro de Kalman

$$\begin{pmatrix} angulo^- \\ deriva^- \end{pmatrix}_K = \begin{pmatrix} 1 & -dt \\ 0 & 1 \end{pmatrix} \begin{pmatrix} angulo \\ deriva \end{pmatrix}_{K-1} + \begin{pmatrix} dt \\ 0 \end{pmatrix} u_K \qquad (69)$$

$$z_K = Hx_k^- + v_K \qquad (70)$$

Dónde:

z_K = velocidad angular proporcionada cada medida por el giróscopo

u_K = velocidad angular dada por el giróscopo

dt = tiempo entre dos medidas

H = vector que une las medidas con el vector de estado. En este caso (1 0)

Angulo = ángulo pitch a estimar, (es lo que realmente queremos estimar)

Deriva = deriva del giróscopo, parte del vector de estado.

4.7.3.6.1. Implementación del Filtro de Kalman del giróscopo y el acelerómetro

Dado el algoritmo de Kalman presentado en el apartado anterior, se tiene el siguiente desarrollo del Filtro para su implementación en la placa.

- Etapa de predicción

Tenemos en este caso: $\hat{x}_K^- = A\hat{x}_{K-1} + Bu_K$

Y según el modelo anterior:

$$\begin{pmatrix} angulo^- \\ deriva^- \end{pmatrix}_K = \begin{pmatrix} 1 & -dt \\ 0 & 1 \end{pmatrix} \begin{pmatrix} angulo \\ deriva \end{pmatrix}_{K-1} + \begin{pmatrix} dt \\ 0 \end{pmatrix} u_K \quad (71)$$

$$donde\ A = \begin{pmatrix} 1 & -dt \\ 0 & 1 \end{pmatrix} \Big| B = \begin{pmatrix} dt \\ 0 \end{pmatrix} \Big| \hat{x}_K = \begin{pmatrix} angulo \\ deriva \end{pmatrix}_K$$

Con lo cual la implementación en la placa estaría dada por

$$\hat{x}[0]_K^- = \hat{x}[0]_{K-1} - \hat{x}[1]_{K-1}dt + u_K dt \quad (72)$$

$$\hat{x}[1]_K^- = \hat{x}[1]_{K-1} \quad (73)$$

Recordemos que el subíndice *k* representa el valor de la magnitud en la interacción actual, *k-1* representa el valor de la magnitud en la interacción anterior del Filtro. El superíndice "-" representa que es la magnitud estimada en la etapa de predicción, es decir que falta la etapa de corrección para obtener el valor a la salida del filtro. El circunflejo "^" indica que el valor de la magnitud es una magnitud estimada.

Vemos como en este caso estamos tratando con vectores de dos componentes, (componentes del vector de estados del Filtro)

Dado:

$$P_K^- = A \cdot P_{K-1} \cdot A^T + Q \quad (74)$$

con $Q = \begin{pmatrix} 0.001 & 0 \\ 0 & 0.003 \end{pmatrix}$

Donde Q[0][0] está relacionado con el error en el modelo del cálculo del ángulo Pitch y el Q[1][1] está relacionado con el modelo que hemos supuesto para la deriva del giróscopo.

En función del sistema se tiene:

$$P_K^- = \begin{pmatrix}1 & -dt\\ 0 & 1\end{pmatrix}\begin{pmatrix}P_{K-1}(0,0) & P_{K-1}(0,1)\\ P_{K-1}(1,0) & P_{K-1}(1,1)\end{pmatrix}\begin{pmatrix}1 & 0\\ -dt & 1\end{pmatrix} + \begin{pmatrix}Q(0,0) & Q(0,1)\\ Q(1,0) & Q(1,1)\end{pmatrix} \rightarrow$$

$$\rightarrow \begin{pmatrix}P_{K-1}(0,0) - P_{K-1}(1,0)dt & P_{K-1}(0,1) - P_{K-1}(1,1)dt\\ P_{K-1}(1,0) & P_{K-1}(1,1)\end{pmatrix}\begin{pmatrix}A & 0\\ -dt & 1\end{pmatrix} + (Q) \rightarrow$$

$$\rightarrow \begin{pmatrix}P_{K-1}(0,0) - P_{K-1}(1,0)dt - P_{K-1}(0,1)dt + P_{K-1}(1,1)dt^2 & P_{K-1}(0,1) - P_{K-1}(1,1)dt\\ P_{K-1}(1,0) - dt \cdot P_{(K-1)}(1,1) & P_{K-1}(1,1)\end{pmatrix}$$

$$P_K^-(0,0) = P_{K-1}(0,0) - P_{K-1}(1,0)dt - P_{K-1}(0,1)dt + P_{K-1}(1,1)dt^2 + Q(0,0) \quad (75)$$

$$P_K^-(0,1) = P_{K-1}(0,1) - P_{K-1}(1,1)dt + Q(0,1) \quad (76)$$

$$P_K^-(1,0) = P_{K-1}(1,0) - P_{K-1}(1,1)dt + Q(1,0) \quad (77)$$

$$P_K^-(1,1) = P_{K-1}(1,1) + Q(1,1) \quad (78)$$

- Etapa de Corrección

Recordando que la ganancia de Kalman viene dado por:

$$K_K = P_K^- H^T (H\, P_K^-\, H^T + R)^{-1} \qquad H = (1 \quad 0), \qquad R = 0.05$$

Donde el valor de R es tomado de la bibliografía consultada.

En este caso P_K (matriz de covarianza del error) será una matriz de 2x2, y H será un vector de 1x2 con valor (1 0).

Desarrollando la ecuación anterior queda:

$$\begin{pmatrix}K[0]\\ K[1]\end{pmatrix} = \frac{\begin{pmatrix}P_K^-(0,0) & P_K^-(0,1)\\ P_K^-(1,0) & P_K^-(0,1)\end{pmatrix}\cdot\begin{pmatrix}1\\ 0\end{pmatrix}}{\left((1 \quad 0)\begin{pmatrix}P_K^-(0,0) & P_K^-(0,1)\\ P_K^-(1,0) & P_K^-(1,1)\end{pmatrix}\begin{pmatrix}1\\ 0\end{pmatrix} + R\right)} = \frac{\begin{pmatrix}P_K^-(0,0)\end{pmatrix}}{P_K^-(0,0) + R} \quad (79)$$

$$K(0) = \frac{\left(P_K^-(0,0)\right)}{P_K^-(0,0) + R} \quad | \quad K(1) = \frac{P_K^-(1,0)}{P_K^-(0,0) + R}$$

Con lo cual la implementación en la placa para el estado corregido estaría dada por:

$$\hat{x}_K = \hat{x}_K^- + K_K(Z_K - H\hat{x}_K^-) \quad (80)$$

$$\begin{pmatrix} x_K(0) \\ x_k(1) \end{pmatrix} = \begin{pmatrix} x_K^-(0) \\ x_K^-(1) \end{pmatrix} + \begin{pmatrix} K(0) \\ K(1) \end{pmatrix} \left[Z_K - (1 \quad 0) \begin{pmatrix} x_K^-(0) \\ x_K^-(1) \end{pmatrix} \right] \quad (81)$$

$$\boldsymbol{x_K(0) = x_K^-(0) + K(0)[Z_K - x_K^-(0)]} \quad (82)$$

$$\boldsymbol{x_K(1) = x_K^-(1) + K(1)[Z_K - x_K^-(0)]} \quad (83)$$

Donde $\hat{x}_K$ representa el vector de estado final estimado por el filtro (el valor que el filtro estima como más exacto- es el valor devuelto por el filtro-) Vector de 2x1

z_K Medida leída (ángulo) por el acelerómetro. Escalar

$\hat{x}_K^-$ representa el vector de estado estimado en la etapa de predicción, calculada anteriormente. Vector de 2x1

K_K La llamada ganancia de Kalman. Es un vector de 2x1

Por otra parte, hay que recalcular la matriz de covarianza del error, que según vimos viene dada por la expresión:

$$P_K = (I - K_K H)P_K^- \quad (84)$$

Donde

$$I = Representa\ la\ matriz\ \ identidad\ de\ 2x2.$$

Teniendo esto en cuenta y según el modelo que hemos definido obtenemos:

$$\begin{pmatrix} P_K(0,0) & P_K(0,1) \\ P_K(1,0) & P_K(1,1) \end{pmatrix} = \left[\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} - \begin{pmatrix} K(0) \\ K(1) \end{pmatrix} (1,0) \right] \begin{pmatrix} P_K^-(0,0) & P_K^-(0,1) \\ P_K^-(1,0) & P_K^-(1,1) \end{pmatrix} =$$

$$= \left[\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} - \begin{pmatrix} K(0) & 0 \\ K(1) & 0 \end{pmatrix} \right] \begin{pmatrix} P_K^-(0,0) & P_K^-(0,1) \\ P_K^-(1,0) & P_K^-(1,1) \end{pmatrix} =$$

$$= \left[\begin{pmatrix} 1 - K(0) & 0 \\ -K(1) & 1 \end{pmatrix} \right] \begin{pmatrix} P_K^-(0,0) & P_K^-(0,1) \\ P_K^-(1,0) & P_K^-(1,1) \end{pmatrix} \rightarrow$$

$$P_K(0,0) = \big(1 - K(0)\big)P_K^-(0,0) \quad (85)$$

$$P_K(0,1) = \big(1 - K(0)\big)P_K^-(0,1) \quad (86)$$

$$P_K(1,0) = -K(1) \cdot P_K^-(0,0) + P_K^-(1,0) \quad (87)$$

$$P_K(0,0) = -K(1) \cdot P_K^-(0,1) + P_K^-(1,1) \quad (88)$$

Dónde:

P_K matriz de covarianza actual tras el Filtro

K_k ganancia de Kalman.

P_K^- matriz de covarianza calculada en la etapa de predicción.

Finalmente recordemos que tenemos que guardar los datos actuales (subíndice *k*) como los valores de las magnitudes de la interacción anterior para la próxima interacción, es decir hay que copiar los valores actuales como los anteriores antes de salir de la función

4.7.3.7. Algoritmo de nivelación de la plataforma utilizando únicamente los datos proporcionados por el acelerómetro

En este caso se llevará a cabo la nivelación de la plataforma únicamente teniendo en cuenta los datos aportados por el acelerómetro.

La función que realiza la nivelación es *void Nivelar_Acelerometro(void)*, se ha implementado en el fichero *medidas.c* que será llamada desde el programa principal. El diagrama de flujo que sigue es

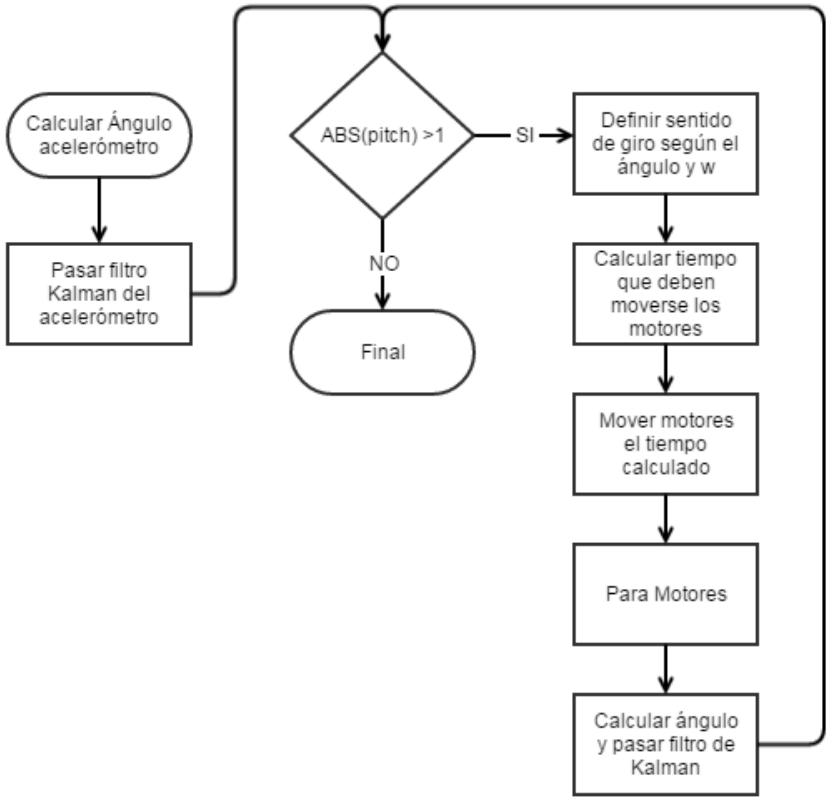


Diagrama de bloques 9 - Algoritmo de nivelación usando únicamente el acelerómetro

En primer lugar leemos el ángulo inicial mediante los datos aportados por el acelerómetro. Posteriormente se pasa ese dato a la función que nos estimará el ángulo utilizando el filtro de Kalman, teniendo en cuenta que dicho ángulo debe ser una constante (la plataforma no se está moviendo). Para ello se llama a una nueva función. *double pasarKalmanAcelerometro(double angulo)* En esta función se inicializa el filtro con el ángulo inicial medido, a continuación se invoca al filtro de Kalman para el acelerómetro tomando 5 medidas para estimar una magnitud constante. Se podría realizar un estudio a fin de estimar las medidas mínimas para la convergencia de la magnitud, pero queda fuera del ámbito de este trabajo su determinación. La función invocada sería

```
double pasarKalmanAcelerometro(double angulo)
{
  double pitchF,r,pitch;
  pitch=angulo;
  // pasamos por el filtro de kalman , con 5 medidas suponemos que c
     inicializacionKalmanAcelerometro(pitch,1.00);
     for(int j=1;j<5;j++){

       // volvemos a leer nuevas entradas
       calculoAngulos(&pitch, &r);


       pitchF=filtroKalmanAcelerometro(pitch);
     }
     return pitchF;
```

Como vemos el número de medidas tomadas para la convergencia es de cinco medidas. Dicho número ha sido tomado de tal forma que nos proporcione un dato aceptable y el consumo de tiempo sea reducido.

Tras estimar el cálculo del ángulo actual se calcula el tiempo y en qué sentido deberá girar la plataforma para nivelase, para ello se ha calculado previamente y de forma externa la velocidad angular de subida y la velocidad angular de bajada.

Se activan los motores durante el tiempo calculado y a continuación se detienen (plataforma estática) y se vuelve a calcular el ángulo repitiendo todo el proceso mientras el ángulo se sitúe entre [-1º, +1º].

4.7.3.8. Algoritmo de nivelación de la plataforma utilizando únicamente los datos proporcionados por el giróscopo

En este caso se llevará a cabo la nivelación de la plataforma únicamente teniendo en cuenta los datos aportados por el giróscopo.

La función que realiza la nivelación es *void Nivelar_Giroscopo(void),* se ha implementado en el fichero *medidas.c* que será llamada desde el programa principal. El diagrama de flujo que sigue es:

Diagrama de bloques 10 - Algoritmo de nivelación usando únicamente el giróscopo

Empezamos calculando el ángulo inicial por medio del acelerómetro (la plataforma está estática por lo que el valor calculado mediante el acelerómetro será fiable), tras verificar que no está dentro de los límites aceptables [-1, +1] inicializamos las diferentes magnitudes que se utilizarán posteriormente: según el signo del ángulo se define un sentido de giro, se define una velocidad angular inicial calculada de forma

externa al programa (nos servirá para inicializar el filtro de Kalman para el giróscopo), se inicializa el tiempo inicial y finalmente se inicializa el filtro de Kalman para el giróscopo.

Tras la inicialización entramos en un bucle que se accederá al interior solo si el ángulo está fuera de los límites aceptables comentados anteriormente (la primera iteración siempre entrará). Si el ángulo está fuera de los límites, es decir, necesita de nivelación, se inicializa el sentido de giro (según signo del ángulo), se define velocidad angular inicial, se verifica si ha cambiado de sentido (ya que si ha cambiado de sentido se deben para motores), se inicializa de nuevo el filtro de Kalman para el giróscopo con la velocidad angular anterior, y se iguala el giro_anterior a giro_actual.

A continuación verificamos si el tiempo inicial es cero. En ese caso los motores estarían parados, y si es diferente de cero los motores estarían en marcha y se deberá medir el tiempo transcurrido desde la última medida. En caso de que el tiempo inicial sea cero, deberá inicializarse el tiempo inicial con la función HAL_GetStick() y enceder los motores, en caso que no sea cero no haría falta realizar estas acciones (los motores ya están en marcha y el tiempo está contándose).

Verificado todo el proceso se pasa a calcular el ángulo con los datos proporcionados por el giróscopo, lo enseñamos por pantalla y volvemos al inicio del bucle, verificando si el ángulo medido está dentro de los límites aceptables. En caso de que ya esté dentro de estos límites se paran los motores y se devuelve el control al programa principal.

Parte de su código se muestra a continuación

```
 while (fabs(pitch)>1)
{

  giro=( pitch>0 ? 1 :2 );
  w=(pitch>0 ? WB :WS);
  if(giro!=giro_ant){  //cambiamos de sentido de giro
       paraMotor();
       inicializacionKalmanGiroscopo(w);
       giro_ant=giro;
       tiempo1=0;
       HAL_Delay(100);
  }

  if( !tiempo1)
  {
    tiempo1=HAL_GetTick();

  moverMotor(giro,VELOCIDAD);
  }

    pitch=CalculoAnguloGiroscopo(pitch);
  i++;
  printf("\n\rAngulo ,medida %ld:%lf (grados)\n",i,pitch);
}
  paraMotor();

}
```

Vemos como se llama a la función *CalculoAnguloGiroscopo(pitch)* para que nos devuelva el nuevo ángulo transcurrido el tiempo entre dos medidas.

4.7.3.9. Algoritmo de nivelación de la plataforma utilizando los datos proporcionados por el acelerómetro y el giróscopo

En este caso se llevará a cabo la nivelación de la plataforma realizando la fusión de los datos proporcionados por giróscopo y acelerómetro. La función que realiza la nivelación es *void Nivelar_Giroscopo_Acelerometro(void)*. Se ha implementado en el fichero *medidas.c* y será llamada desde el programa principal. El diagrama de flujo que sigue es

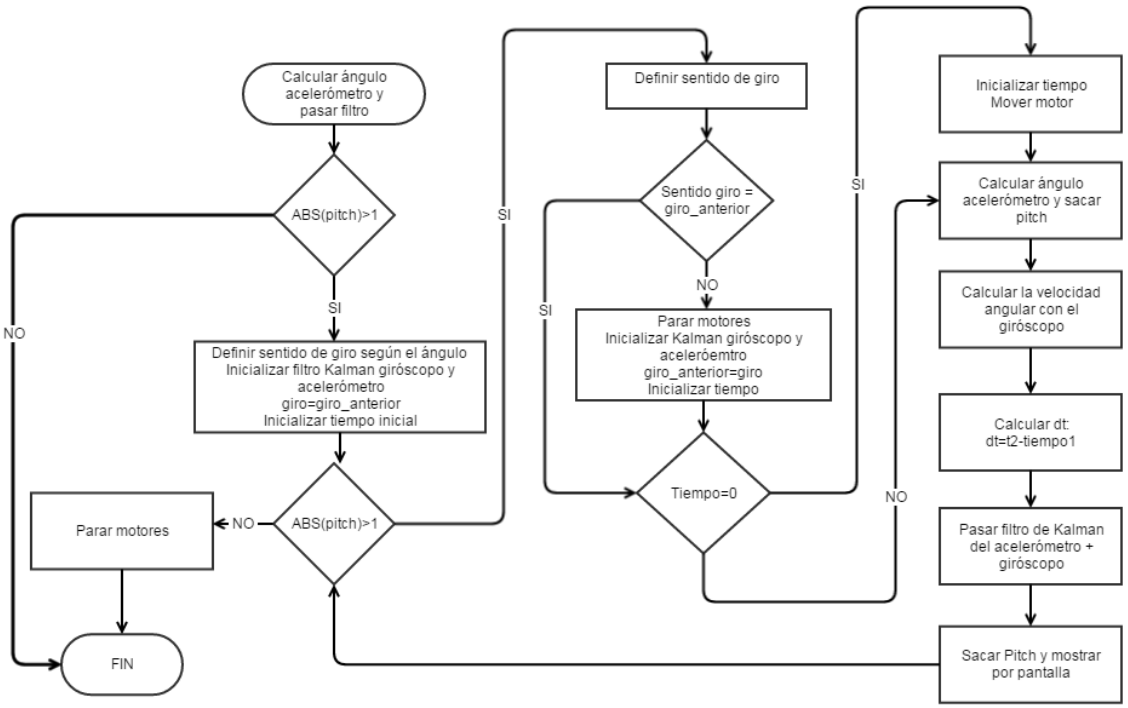


Diagrama de bloques 11 - Algoritmo de nivelación usando el acelerómetro y el giróscopo

Como vemos el diagrama es muy parecido a la nivelación mediante el giróscopo, aunque con algunas diferencias:

En primer lugar, a la inicialización del filtro de Kalman, en este caso, se le debe pasar el ángulo y no la velocidad angular (en este caso estimaremos el ángulo).

En segundo lugar, es necesario estimar tanto el ángulo medido por el acelerómetro como la velocidad angular medida por el giróscopo. El ángulo se pasará al filtro de Kalman como medida de referencia y la velocidad angular como entrada al sistema. También se necesita pasar al filtro, en este caso, el tiempo transcurrido.

Con todo ello estimamos el ángulo con el filtro de Kalman elaborado expresamente para esta situación. Este ángulo es el que se evalúa para verificar si está dentro de los límites establecidos. En este caso no hay función que calcula expresamente el ángulo, por lo que es directamente el filtro el que nos proporciona el ángulo recorrido en un determinado tiempo (dt). Por tanto, en este caso es muy importante la medida del tiempo transcurrido.

Parte de código se puede ver en la figura siguiente:

```
if( !tiempo1)
{
  tiempo1=HAL_GetTick();

  moverMotor(giro,VELOCIDAD);
}

calculoAngulos(&pitch, &row);
printf("\n pich %lf",pitch);
Leer_VelocidadAngular_XYZ(pfData);
u=pfData[0];
printf("\n vel angular: %lf",u);
t2=HAL_GetTick();
dt=t2-tiempo1;
tiempo1=t2;
pitch=filtroKalmanAcelerometroGiroscopo( pitch, dt, u);
i++;
printf("\n\rAngulo ,medida %ld:%lf (grados)\n",i,pitch);
//paraMotor();
//while(!leerBoton()){}
  //moverMotor(giro,VELOCIDAD);

paraMotor();
```

Vemos como no llamamos a ninguna función de cálculo del ángulo, a diferencia de los otros dos casos.

Por todo lo demás, como vemos en el diagrama de flujo, es idéntico a lo comentado en el caso de solo utilizar el giróscopo.

5. PRUEBAS Y RESULTADOS

Tras el desarrollo teórico y el montaje físico, se han realizado ciertas pruebas para calcular y comprobar la precisión del prototipo. Dichas pruebas se han basado en dos fases para cada método de medición: por una parte se han realizado pruebas de medición de ángulos ante subidas y bajadas para obtener la precisión en las medidas y así concluir cuál de los tres sistemas será más viable para el trabajo actual, y por otra se han hecho pruebas de nivelación ante diferentes ángulos iniciales para observar la respuesta del sistema y la fiabilidad de éste. A continuación se explican y detallan todas las pruebas realizadas, así como sus resultados y observaciones.

5.1. Medición del ángulo

La primera prueba realizada en cada método de nivelación ha sido la medición de diferentes ángulos tanto en subida como en bajada. Las pruebas han consistido en iniciar el prototipo nivelado en 0º y a continuación realizar un ascenso o descenso en función del intervalo de tiempo indicado. Finalmente, una vez el prototipo ha frenado, se ha comparado el ángulo estimado por el filtro de Kalman con el ángulo real del prototipo. Para medir con precisión este último ángulo se ha utilizado la aplicación "Bubble Level" para Smartphone. Como último paso, se ha realizado una tabla donde se recopilan 12 medidas para poder realizar las comparaciones y obtener, así, la precisión del método utilizado para la medición de ángulos en movimiento.

A continuación se detallan las mediciones obtenidas con los tres tipos de sensores: acelerómetro, giróscopo y acelerómetro + giróscopo.

5.1.1. Medición del ángulo utilizando el acelerómetro

Para esta prueba se ha utilizado la función "AnguloAcelerometro", la cual se encarga de medir el ángulo actual utilizando únicamente el acelerómetro. Esta función lee los datos una vez el prototipo se encuentra estático, por lo que es necesario dejar el brazo del prototipo en una posición concreta y luego llamar a dicha función para medir. Para estas pruebas se han medido los ángulos utilizando escuadra y cartabón, tal y como se ve en la Ilustración 30, ya que era una buena forma de medir unos ángulos conocidos de forma estática.

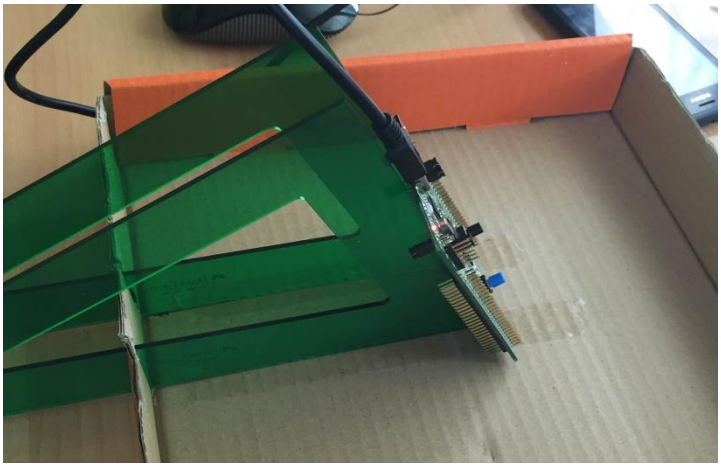


Ilustración 30 - Medición del ángulo utilizando sólo el acelerómetro

Los resultados en la medición de ángulos con acelerómetro fueron los siguientes:

	Angulo original	Angulo medido	Diferencia	Error
1	0.80	0.80	0.00	0.00%
2	0.80	0.70	0.10	12.5%
3	0.80	1.00	0.20	25.00%
4	30.00	29.15	0.85	2,83%
5	30.00	29.08	0.92	3,07%
6	30.00	28.98	1.02	3,40%
7	42.00	42.27	0.27	0,64%
8	42.00	42.37	0.37	0,88%
9	42.00	42.52	0.52	1,24%
10	60.00	60.68	0.68	1,13%
11	60.00	60.23	0.23	0,38%
12	60.00	59.98	0.02	0,03%

Tabla 6 - Ángulos medidos con Acelerómetro

Se puede observar que la diferencia entre el ángulo medido (el ángulo medido por el propio acelerómetro y tratado con el filtro de Kalman) y el ángulo real es muy pequeña, de menos de 1 grado.

Para cuantificar el error se ha utilizado el cálculo del error relativo, cuya fórmula es:

$$error_{relativo} = \frac{|\bar{x} - x|}{\bar{x}} * 100 \quad (89)$$

Siendo $\bar{x}$ el ángulo real del prototipo y x el valor del ángulo medido.

Este error oscila entre el 1 y el 3%, exceptuando cuando el ángulo es muy pequeño (próximo a 0) que una pequeña desviación provoca un gran error. Por tanto podemos concluir que este es un método de medición muy preciso.

Posteriormente, en el apartado 5.1.4. Comparación de resultados, se explicarán las ventajas y desventajas de este método.

5.1.2. Medición del ángulo utilizando el giróscopo

Para la realización de esta prueba se ha utilizado la función de "AnguloGiroscopo", la cual se puede editar para variar el tiempo de movimiento del brazo (con lo cual conseguiremos un grado variable en función de dicho tiempo), y seleccionar un sentido de giro u otro.

Dicha prueba consiste en, partiendo del prototipo nivelado a 0 grados, utilizar la función anteriormente nombrada, la cual se encarga de mover el brazo del prototipo y a su vez realizar medidas periódicas de los grados actuales para, finalmente, proporcionar el valor del ángulo final medido por el giróscopo previo al filtro, y el valor del ángulo una vez ha actuado el filtro de Kalman.

Para está prueba se han realizado dos tomas de datos: la primera toma se ha realizado con el sentido de giro ascendente, y la otra con el sentido descendente. Por lo tanto, tras realizar las pruebas se han obtenido dos tablas de resultados:

	Angulo original	Angulo medido	Diferencia	Error
1	18.20	14.75	3.45	18.96%
2	13.60	10.61	2.99	21.99%
3	8.80	5.55	3.25	36.93%
4	4.40	1.52	2.88	65.45%
5	1.20	-2.11	3.31	275.83%
6	-3.20	-5.45	2.25	70.31%
7	-7.40	-11.78	4.38	59.19%
8	6.60	3.00	3.60	54.55%
9	2.20	-0.08	2.28	103.64%
10	-6.60	-9.92	3.32	50.30%
11	-10.80	-11.29	0.49	-4.54%
12	-14.60	-17.30	2.70	18.49%

Tabla 7 - Ángulos medidos con Giróscopo en Subida

	Angulo original	Angulo medido	Diferencia	Error
1	-29.60	-23.08	-6.52	22.03%
2	-33.60	-27.25	-6.35	18.90%
3	-35.20	-32.97	-2.23	6.34%
4	-37.20	-34.14	-3.06	8.23%
5	-32.80	-28.63	-4.17	12.71%
6	-18.20	-13.64	-4.56	25.05%
7	-2.80	1.74	-4.54	162.14%
8	-35.20	-29.06	-6.14	17.44%
9	-25.00	-19.15	-5.85	23.40%
10	-14.20	-8.88	-5.32	37.46%
11	-3.20	-0.85	-2.35	73.44%
12	6.20	8.66	-2.46	39.68%

Tabla 8 - Ángulos medidos con Giróscopo en Bajada

Como podemos observar, en la primera tabla (medición del ángulo en subida) se puede apreciar que las medidas difieren en torno a los 2 o 3 grados, y los errores obtenidos (calculados igual que en la anterior prueba) son excesivamente altos.

Si observamos la segunda tabla, mediciones en bajada, podremos observar que las diferencias son mayores, en torno a los 4 o 5 grados, y los errores, aunque menores, siguen siendo altos. Importante recordar que, así como las pruebas de ángulos con acelerómetro eran estáticas, en el caso del giróscopo se realizan con el brazo del prototipo en movimiento ya que éste mide la velocidad angular para poder obtener el ángulo.

En el apartado 5.1.4. Comparación de resultados se discutirán y analizarán los resultados obtenidos.

5.1.3. Medición del ángulo utilizando el acelerómetro y el giróscopo

Finalmente, la última prueba de medición de ángulos se trata de la realizada combinando los dos sensores probados anteriormente. En este caso, para el software se utilizará la función "Angulo_Giroscopo_Acelerometro", que tendrá una funcionalidad similar a la utilizada en las pruebas con giróscopo: dicha función se encargará de accionar, en un sentido u otro y con un tiempo ajustable, el brazo robot para variar el ángulo actual, y mientras el brazo está en movimiento los dos sensores tomarán medidas de ángulos y serán tratados por el filtro de Kalman.

En esta prueba también se han realizado dos tomas de datos, una para ángulos en subida y otra para ángulos en bajada. Los resultados han sido los siguientes:

	Angulo original	Angulo medido	Diferencia	Error
1	4.00	3.83	0.17	4.25%
2	17.40	18.07	0.67	3.85%
3	28.60	27.94	0.66	2.31%
4	-4.60	-4.97	0.37	8.04%
5	5.20	5.65	0.45	8.65%
6	15.60	15.82	0.22	1.41%
7	26.00	25.50	0.50	1.92%
8	35.40	35.81	0.41	1.16%
9	44.00	45.40	1.40	3.18%
10	61.00	59.42	1.58	2.59%
11	-27.00	-26.80	0.20	0.74%
12	-12.40	-11.80	0.60	4.84%

Tabla 9 - Ángulos medidos con Giróscopo y Acelerómetro en Subida

	Angulo original	Angulo medido	Diferencia	Error
1	28.00	28.68	0.68	2.43%
2	19.40	19.60	0.20	1.03%
3	11.80	12.00	0.20	1.69%
4	-5.80	-5.54	0.26	4.48%
5	40.60	42.00	1.40	3.45%
6	33.20	33.98	0.78	2.35%
7	25.20	25.67	0.47	1.87%
8	16.80	17.30	0.50	2.98%
9	8.20	8.59	0.39	4.76%
10	-0.80	-0.39	0.41	51.25%
11	-10.80	-9.80	1.00	9.26%
12	-17.80	-17.11	0.69	3.88%

Tabla 10 - Ángulos medidos con Giróscopo y Acelerómetro en Bajada

Como se puede observar, la primera tabla correspondiente a las medidas en subida, nos ha proporcionado unos valores muy precisos respecto a los ángulos reales. Podemos observar que las diferencias entre ambos ángulos están por debajo de 1 en la mayoría de

las medidas. A su vez el error obtenido (calculado de la misma manera que las pruebas anteriores) está comprendido entre el 1 y 3% generalmente.

En el caso de las medidas obtenidas en bajada se puede observar que los resultados son muy similares ya que las diferencias se encuentran por debajo de 1, y los errores entre el 1 y 3%. Hay que remarcar que los datos en bajada tienen un error mayor a los de subida. Todas las conclusiones se comentarán en el apartado 5.1.4. Comparación de resultados.

5.1.4. Comparación de resultados

Tras la realización de todas las pruebas para la obtención de ángulos con los 3 métodos, podemos concluir:

En primer lugar, como se puede observar en los resultados, utilizar el acelerómetro para la medición de ángulos es un método muy preciso y nos proporciona valores muy próximos a la realidad. El gran inconveniente de este método se encuentra en el propio acelerómetro, ya que para obtener resultados precisos y próximos a la realidad debe de tomar medidas en una posición estática. Si se intentan realizar mediciones de ángulos con el acelerómetro mientras éste se mueve, se obtienen valores muy erráticos y poco exactos. Por lo tanto, y como conclusión a este método, se puede decir que el método de medición mediante Acelerómetro y filtro de Kalman es muy preciso pero únicamente para sistemas estáticos y sin vibraciones. En nuestro caso, la finalidad última de éste trabajo consiste en la realización de un nivelador automático, por lo que conlleva el movimiento del brazo del prototipo, excluyendo este método como uno de los posibles. De igual manera, en las pruebas posteriores se realizaran nivelaciones con este método y se discutirán sus resultados.

En segundo lugar tenemos el giróscopo. En este caso los resultados nos han proporcionado unos errores muy altos. Esto es debido a que el giróscopo por si solo proporciona unas medidas muy poco precisas del ángulo actual del sistema, y aún actuando el filtro de Kalman se obtienen valores poco precisos. Hay que añadir que, en toda la documentación consultada para la realización de este trabajo, siempre se remarca que el giróscopo como método de medición es muy poco fiable por sí solo. Por lo que, para concluir las pruebas con este sensor, se puede decir que no es un método fiable y por lo tanto no es una opción viable para nuestro desarrollo. Al igual que con el acelerómetro, en las posteriores pruebas se realizarán nivelaciones con el giróscopo y analizaremos sus resultados.

Finalmente, las últimas pruebas realizadas han sido con el acelerómetro y giróscopo trabajando en conjunto. Ante los resultados obtenidos podemos concluir que

es el método más fiable y con mejor precisión de los utilizados, ya que las pruebas realizadas han sido todas en movimiento y se ha obtenido unos resultados precisos y fieles a la realidad. Por lo tanto, ante las primeras pruebas realizadas, se puede concluir que el método a utilizar en el prototipo final será el basado en acelerómetro y giróscopo con filtro de Kalman.

5.2. Pruebas de nivelación

Las últimas pruebas realizadas han sido las pruebas de nivelación, en las cuales el brazo del prototipo deberá subir o bajar un cierto ángulo, y posteriormente corregir dicho ángulo hasta nivelarse en 0 grados. En estas pruebas, los resultados se han dividido en 4 columnas: Por una parte tenemos las medidas devueltas por los sensores tras pasar por el filtro de Kalman, las medidas reales, la diferencia de la medida real respecto de 0 y finalmente el número de intentos que consiste en el número de veces que el sistema ha intentado la nivelación completa hasta conseguirlo.

A continuación se muestran los diversos resultados obtenidos mediante los tres métodos utilizados.

5.2.1. Nivelación utilizando el acelerómetro

En primer lugar se han realizado las pruebas de nivelación utilizando el acelerómetro. En este caso se han realizado dos mediciones, nivelación en subida y nivelación en bajada.

Hay que tener en cuenta que tal y como se ha explicado anteriormente, las medidas del acelerómetro únicamente son fiables si el sistema es estático, con lo que para realizar la nivelación del prototipo se calcula el ángulo inicial y sabiendo la velocidad angular de antemano se calcula el tiempo que debe estar encendido el motor. Una vez transcurrido este tiempo, el sistema se detiene, vuelve a medir y vuelve a mover el prototipo si es necesario.

Los resultados obtenidos con este método han sido los siguientes:

	Medida Sensor	Medida Exterior	Diferencia	intentos
1	0.36	-0.60	0.60	2
2	-0.33	-0.40	0.40	2
3	-0.42	-0.40	0.40	2
4	-0.13	-0.60	0.60	2
5	-0.29	-0.60	0.60	2
6	0.09	-0.80	0.80	4
7	0.54	-0.40	0.40	2
8	-0.29	-0.40	0.40	3
9	-0.88	-1.80	1.80	2
10	-0.44	-1.20	1.20	3

Tabla 11 - Nivelación en subida utilizando Acelerómetro

	Medida Sensor	Medida Exterior	Diferencia	intentos
1	0.16	-0.40	0.40	2
2	0.06	-0.20	0.20	2
3	-0.36	0.40	0.40	2
4	-0.92	-1.20	1.20	2
5	-0.18	-0.60	0.60	2
6	-0.31	-0.40	0.40	2
7	-0.03	-1.00	1.00	3
8	0.07	-0.80	0.80	2
9	0.97	-0.20	0.20	2
10	0.23	-0.60	0.60	2

Tabla 12 - Nivelación en bajada utilizando Acelerómetro

Observando la primera tabla tenemos los resultados de la nivelación en subida. Podemos apreciar que las diferencias entre la medida real y la calculada no es grande y a su vez tenemos unos errores, en general, bajos. El parámetro que limita este método es el número de intentos realizado para conseguir la nivelación.

Como se puede apreciar, en ninguna de las medidas se consigue la nivelación en un intento, y encontramos medidas que necesitan hasta cuatro intentos para obtener una nivelación aceptable.

Si analizamos la segunda tabla nos encontramos con unos resultados idénticos a los anteriores. Se tiene una precisión buena, con un error en general bajo, pero con la necesidad de realizar varios intentos para conseguir la nivelación. Ante estos resultados, posteriormente se analizarán y tratarán dichos resultados en común con el resto de métodos.

5.2.2. Nivelación utilizando el giróscopo

En segundo lugar se han realizado pruebas de nivelación con el giróscopo, realizando también dos mediciones: Nivelación en subida y nivelación en bajada.

Los resultados obtenidos han sido:

	Medida Sensor	Medida Exterior	Diferencia	intentos
1	-0.99	-2.40	1.41	1
2	-0.99	-2.60	1.61	1
3	-0.99	-3.00	2.01	1
4	-0.98	-2.00	1.02	1
5	-0.99	-1.80	0.81	1
6	-0.99	-0.80	0.19	1
7	-0.99	0.00	0.99	1
8	-1.03	-0.04	0.63	1
9	-0.99	-1.00	1.00	1
10	-0.99	-0.06	0.66	1

Tabla 13 - Nivelación en subida utilizando Giróscopo

	Medida Sensor	Medida Exterior	Diferencia	intentos
1	1.06	-4.20	4.20	1
2	1.00	-5.40	5.40	1
3	0.99	-4.60	4.60	1
4	0.99	-4.40	4.40	1
5	0.99	-4.20	4.20	1
6	0.98	-4.60	4.60	1
7	0.98	-5.20	5.20	1
8	0.99	-4.20	4.20	1
9	0.99	-5.00	5.00	1
10	0.98	-5.40	5.40	1

Tabla 14 - Nivelación en bajada utilizando Giróscopo

Analizando la primera tabla se puede observar que los resultados proporcionan unos errores altos y una nivelación en un único intento, lo que no son resultados muy positivos.

Si se observa la segunda tabla se puede apreciar que los resultados son peores. En este caso la precisión del sensor es muy baja ya que está obteniendo diferencias en los ángulos de 4 grados o superiores, por lo que el error supera el máximo permitido (dos grados de margen). De esta forma los segundos resultados son negativos ya que aportan poca fiabilidad y precisión al sistema. En la comparación de resultados posterior se analizarán los datos obtenidos.

5.2.3. Nivelación utilizando el acelerómetro y el giróscopo

Por último se han realizado las pruebas al método del acelerómetro y giróscopo. En este caso, al igual que los otros métodos, también se han realizado dos mediciones: nivelación en subida y nivelación en bajada.

Los resultados obtenidos han sido los siguientes:

	Medida Sensor	Medida Exterior	Diferencia	intentos
1	-0.61	-1.00	1.00	1
2	-0.48	-0.80	0.80	1
3	-0.68	-0.20	0.20	1
4	-0.72	-0.60	0.60	1
5	-0.44	-0.60	0.60	1
6	-0.46	-0.80	0.80	1
7	-0.45	-0.80	0.80	1
8	-0.24	-0.60	0.60	1
9	-0.52	-0.80	0.80	1
10	-0.41	-0.80	0.80	1

Tabla 15 - Nivelación en subida utilizando Giróscopo y Acelerómetro

	Medida Sensor	Medida Exterior	Diferencia	intentos
1	0.35	0.80	0.80	1
2	0.67	1.00	1.00	1
3	0.30	-0.60	0.60	1
4	0.36	-0.20	0.20	1
5	0.38	-0.20	0.20	1
6	0.44	0.60	0.60	1
7	0.43	0.60	0.60	1
8	0.42	0.20	0.20	1
9	0.63	0.40	0.40	1
10	0.24	0.60	0.60	1

Tabla 16 - Nivelación en bajada utilizando Giróscopo y Acelerómetro

Observando la primera tabla de resultados, la correspondiente a la nivelación en subida, podemos concluir que los datos obtenidos han sido muy precisos, con poco margen de error y rápidos ya que se ha conseguido nivelar en un único intento.

Analizando la segunda tabla concluimos con la misma opinión que los datos anteriores, este método ha proporcionado unos resultados precisos, rápidos y fiables.

5.2.4. Comparación de resultados

Una vez realizadas todas las pruebas de nivelación, se contrastarán los resultados:

En primer lugar se tienen los resultados obtenidos mediante la nivelación con acelerómetro y filtro de Kalman. Se han obtenido unos resultados finales precisos pero con una velocidad de nivelación baja y una precisión relativa baja. Como se concluyó en las medidas de ángulo con este método, el acelerómetro constituye un buen sistema de medición para entornos estáticos, pero en movimiento genera muchas medidas erróneas lo que le conduce a ser poco preciso en un periodo corto de tiempo. Tras varios intentos consigue nivelarse con unos resultados aceptables, pero esa poca precisión inicial lo hace un método poco viable.

En segundo lugar tenemos los resultados obtenidos con el giróscopo. En subida proporcionan unos valores negativos ya que, en mitad de las ocasiones, los resultados presentan errores por encima de lo permitido. Los resultados en bajada son peores ya que presentan errores mucho mayores de lo permitido. Esta diferencia entre las medidas en subida y en bajada está causada por el motor del prototipo. En el caso de la subida tiene un movimiento más lento ya que, al no ser un motor de precisión, en la subida pierde fuerza. Al tener un movimiento más lento y por lo tanto una velocidad angular menor, permite un mayor número de mediciones, lo que resulta en unos valores más próximos a la realidad. Igualmente, se puede observar que la utilización del giróscopo con filtro de Kalman como método de medición y nivelación no es válido ya que nos aporta unos resultados fuera de los rangos permitidos,

Finalmente analizamos los resultados obtenidos por el giróscopo y acelerómetro con filtro de Kalman. Se puede observar que son unos resultados muy precisos, con unos errores dentro de lo permitido y unas diferencias respecto a la realidad muy pequeñas. Como ya se ha concluido en las pruebas de medición de ángulos, el método de giróscopo y acelerómetro es el que mejores resultados aporta para la finalidad que se busca en este trabajo. Proporciona muy buenos resultados en sistemas móviles, con un rango de error muy pequeño y una precisión muy alta. Por lo que, tras las pruebas, queda claro que el método adecuado para un nivelador automático es la utilización del giróscopo y acelerómetro en conjunto con un filtro de Kalman.

6. CONCLUSIONES Y TRABAJOS FUTUROS

Para concluir el presente trabajo, se presentarán las conclusiones obtenidas tras el estudio e implementación de diversos métodos para conseguir un nivelador automático, y también se presentarán diversos trabajos que se realizarán en un futuro siguiendo la línea de este trabajo y el estudio correspondiente.

6.1. Conclusiones

Como conclusiones del estudio e implementación de los tres métodos elegidos, nivelación mediante acelerómetro y filtro de Kalman, nivelación mediante giróscopo y filtro de Kalman y nivelación mediante giróscopo, acelerómetro y filtro de Kalman, se puede concluir que el mejor método para la realización correcta de un nivelador automático es la utilización de la fusión sensorial del giróscopo y el acelerómetro trabajando en conjunto y la implementación de un filtro del Kalman como proceso de control.

Esta conclusión viene respaldada por las pruebas realizadas, en las que este sistema ha proporcionado unos datos muy precisos y dentro de los límites establecidos, y a su vez no ha supuesto una gran inversión económica.

También se concluye que la tarjeta STM32F3 es la mejor de su familia para la implementación de dicho nivelador ya que cuenta con los sensores necesarios, y estos tienen unas buenas especificaciones técnicas.

Otra conclusión obtenida en el desarrollo del presente trabajo ha sido la elección del sistema de control. Tras estudiar las ventajas e inconvenientes se ha podido concluir que el filtro de Kalman representa una buena alternativa para el control y tratamiento de los datos obtenidos de los sensores. Dicho filtro proporciona unos resultados muy precisos, es capaz de trabajar ante entornos cambiantes y con diversidad de errores y existe una gran cantidad de información al respecto, lo que hacen viable su estudio e implementación. Además, nos permite estimar variables cuya medida no es accesible para el usuario como puede ser el error de deriva del giróscopo. Como punto negativo se ha observado que es necesario realizar la nivelación a velocidades angulares bajas ya que a velocidades altas, el sistema presenta un periodo de aceleración elevado en el cual la velocidad angular no es constante, por lo que invalida el modelo del filtro escogido.

Respecto a la estructura empleada, existen mejores alternativas para desarrollar un sistema más preciso y fiable, pero al tratarse, el presente trabajo, del desarrollo de un prototipo de nivelador automático para la utilización en la construcción, se ha decidido realizar un montaje económico, capaz de recrear las necesidades existentes en la construcción y que pueda servir como base para futuros proyectos.

6.2. Trabajos futuros

Tras la realización del presente trabajo, se ha demostrado que la tarjeta STM32F3 Discovery es apta para la realización de proyectos de auto nivelación en un eje, aportando una buena precisión. Ante esta situación se abre un gran abanico de posibilidades a desarrollar en un futuro:

El principal trabajo a realizar en un futuro será el de diseñar una estructura que aporte una mayor robustez y exactitud, con un motor de precisión. A si mismo dicha estructura se deberá de diseñar con dos puntos de apoyo para aportar una mayor estabilidad. También se debería añadir un panel de control para poder seleccionar los modos de nivelación, precisión y medida de ángulos que queremos utilizar.

Otro trabajo a desarrollar en la línea del presente consta de la realización de un nivelador automático capaz de nivelar dos ejes. En este caso, este desarrollo conllevaría rehacer el filtro de Kalman y parte de la programación ya que actualmente, únicamente se contempla la actuación del sistema en un único eje, el eje Y.

Como se puede ver, el presente trabajo puede servir como punto de partida para la realización de otros trabajos que conllevan una mayor carga técnica, mecánica y electrónica.

7. BIBLIOGRAFIA

[1] STMicroelectronics, «Discovery kit with STM32F303VC MCU» DM00063389.

[2] STMicroelectronics, « User Manual», DM00063382.

[3] STMicroelectronics, «Datasheet L3GD20», DM00036465.

[4] STMicroelectronics, «Datasheet LSM303DLHC», DM00027543.

[5] S. Marqués, «Desarrollo de software de bajo nivel para un brazo robot portátil», ETSID (UPV), Valencia, 2016.

[6] D. Pozo, «Diseño y construcción de una plataforma didáctica para medir ángulos de inclinación usando sensores inerciales como acelerómetro y giróscopo», Escuela Politécnica Nacional, Quito, 2010.

[7] O. Terrejón, «Diseño, fabricación y caracterización de un sensor de caudal para aplicaciones PCB_MEMS», ETSI (Universidad de Sevilla), Sevilla, 2011.

[8] J.V. Capella y A.Perles «Guía de iniciación al kit de evaluación st STM32F4 Discovery», UPV, Valencia, 2012.

[9] STMicroelectronics, «Datasheet STM32F303xC», DM00058181.

[10] D.P. Vigouroux, «Implementación de unidad de mediciones inerciales (IMU) para robótica utilizando filtro de Kalman», Universidad Simón Bolívar, 2010

[11] B. Welch, «An Introduction to the Kalman Filter». EEUU: Department of Computer Science University of North Carolina, July 2006.

[12] Simon D. «Kalman Filtering», Junio 2001.

[13] G. Ferrer, «Integración Kalman de sensores inerciales INS con GPS en un UAV», UPC, 2009.

[14] InvenSense, «MEMS Gyroscope technology».

[15] A. Solera, «El filtro de Kalman», Banco Central de Costa Rica, Julio 2003.

[16] tompyckebe, «Kalman filtering of IMU data», Mayo 2006.

[17] P. Maybeck, «Introduction from Stochastic Models, estimation and control, volumen 1», Department of electrical engineering air forcé institute of technology Wright-Patterson air forcé base, OHIO 1979.

8. ILUSTACIONES

9. TABLAS

10. DIAGRAMAS DE BLOQUES

11. ANEXOS

11.1.1. Código Fichero main.c

```
/**
  ******************************************************************************
* File Name          : main.c
* Description        : PROGRAMA FINAL DE GRADO (NIVELACION DE UNA PLATAFORMA CON UN  GRADO DE LIBERTAD)
                       Programa principal,
  * Author           : Ivan
  ******************************************************************************
  * Programa principal desde el cual se llamarán a las funciones principales, tanto  de
  * inicialización como las funciones objetivo del proyecto,.
  *
  *
  ******************************************************************************
  */
/*  Includes ------------------------------------------------------------------*/
#include "stm32f3xx_hal.h"
#include "lsm303dlhc.h"
#include "motores.h"


/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private variables  ---------------------------------------------------------*/
extern I2C_HandleTypeDef hi2c1;    //bus I2C

extern SPI_HandleTypeDef hspi1;    // bus SPI

extern TIM_HandleTypeDef htim4;    // timer utilizado

/* USER CODE BEGIN PV */
/* Private variables  ---------------------------------------------------------*/

/* USER CODE END PV */
void Leer_Aceleracion_XYZ(int16_t* );
HAL_StatusTypeDef  InicializarAcelerometro(void);
void  InicializarSensores(void);
void Leer_VelocidadAngular_XYZ(float  *);
HAL_StatusTypeDef InicializarGiroscopo(void);

/* Private function prototypes  -----------------------------------------------*/
void  SystemClock_Config(void);
void  Error_Handler(void);
static void  MX_GPIO_Init(void);
extern void  MX_I2C1_Init(void);
extern void  MX_SPI1_Init(void);
static void MX_TIM4_Init(void);

void  HAL_TIM_MspPostInit(TIM_HandleTypeDef  *htim);

int  leerBoton(void);
void calculoAngulos(double *p, double  *r);
void  tomarBiasAcelerometro(void);
void  tomarBiasGiroscopo(void);
void   AnguloAcelerometro(void);
void  AnguloGiroscopo(void);
void Angulo_Giroscopo_Acelerometro(void);

void  moverMotor(int);
void  paraMotor(void);
void subirMotor(int  );
void bajarMotor(int );

void  Nivelar_Acelerometro(void);
void  Nivelar_Giroscopo(void);
void  Nivelar_Giroscopo_Acelerometro(void);
/* USER CODE BEGIN PFP */
/* Private function  prototypes -----------------------------------------------*/

/* USER CODE END  PFP */

/* USER CODE BEGIN  0 */

/* USER CODE END  0 */
```

```
76  int main(void)
77  {
78
79    /* USER CODE BEgin  */
80
81
82    /* USER CODE END 1 */
83
84    /*  MCU Configuration--------------------------------------------------------*/
85
86    /* Reset of all peripherals, Initializes the Flash interface and the Systick.  */
87    HAL_Init();
88
89    /* Configuramos el sistema de reloj  */
90    SystemClock_Config();
91
92    /* Inicializamos todos los periféricos  */
93    MX_GPIO_Init();
94    MX_I2C1_Init();
95    MX_SPI1_Init();
96    MX_TIM4_Init();
97
98    /* USER CODE BEGIN 2  */
99  //Activamos los pines de alimentación del Driver
100
101 HAL_GPIO_WritePin(GPIOB,GPIO_PIN_5,GPIO_PIN_SET);
102 HAL_GPIO_WritePin(GPIOB,GPIO_PIN_9,GPIO_PIN_SET);
103
104 // ACTIVAMOS  EL PWM  PD14 y PD15 (CANALES 3 Y 4 DEL  TIMER 4
105
106   HAL_TIM_PWM_Start(&htim4,TIM_CHANNEL_4);    // ACTIVAMOS  EL PWM  PD15  led azul
107   HAL_TIM_PWM_Start(&htim4,TIM_CHANNEL_3);    // ACTIVAMOS  EL PWM  PD14  led azul
108   /* USER CODE END 2 */
109
110
111
112
113
114   printf("\n\rPulse el boton de usuario para Empezar  ");
115     while(!leerBoton()){}   //esperamos a que se pulse el  boton.
116
117     InicializarSensores(); // inicializamos  acelerómetro y  giróscopo.
118
119             // descomentar la función que se desee probar, comentar  el resto
120
121 /**********funciones de puesta a  punto****************/
122
123         subirMotor(6000);
124       //bajarMotor(10000);
125       HAL_Delay(2000);
126       /***********  funciones de calculos de ángulos  **********/
127
128       //AnguloAcelerometro();
129       //AnguloGiroscopo();
130       //Angulo_Giroscopo_Acelerometro();
131
132       /********** funciones de nivelación  **************/
133
134       //Nivelar_Acelerometro();
135       //Nivelar_Giroscopo();
136         Nivelar_Giroscopo_Acelerometro();
137
138
139
140       //Nivelar_Acelerometro();
141
142
143
144
145
146   /* Infinite loop */
147   /* USER CODE BEGIN WHILE */
148   while (1)
149   {
150   /* USER CODE END WHILE */
151
```

```
152     /* USER CODE BEGIN 3 */
153
154
155
156
157     }
158     /* USER CODE END 3 */
159
160   }
161
162   /** System Clock Configuration
163   */
164   void SystemClock_Config(void)
165   {
166
167     RCC_OscInitTypeDef RCC_OscInitStruct;
168     RCC_ClkInitTypeDef RCC_ClkInitStruct;
169     RCC_PeriphCLKInitTypeDef PeriphClkInit;
170
171     RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
172     RCC_OscInitStruct.HSIState = RCC_HSI_ON;
173     RCC_OscInitStruct.HSICalibrationValue = 16;
174     RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
175     RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
176     RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL12;
177     if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
178     {
179       Error_Handler();
180     }
181
182     RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
183                                 |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
184     RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
185     RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
186     RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
187     RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
188     if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1) != HAL_OK)
189     {
190       Error_Handler();
191     }
192
193     PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_I2C1;
194     PeriphClkInit.I2c1ClockSelection = RCC_I2C1CLKSOURCE_HSI;
195     if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
196     {
197       Error_Handler();
198     }
199
200     HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);
201
202     HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);
203
204     /* SysTick_IRQn interrupt configuration */
205     HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
206   }
207
208   /* I2C1 init function */
209    void MX_I2C1_Init(void)
210   {
211
212     hi2c1.Instance = I2C1;
213     hi2c1.Init.Timing = 0x2000090E;
214     hi2c1.Init.OwnAddress1 = ACC_I2C_ADDRESS;
215     hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
216     hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
217     hi2c1.Init.OwnAddress2 = 0;
218     hi2c1.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
219     hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
220     hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
221     if (HAL_I2C_Init(&hi2c1) != HAL_OK)
222     {
223       Error_Handler();
224     }
225
226       /**Configure Analogue filter
227       */
```

```
228     if (HAL_I2CEx_ConfigAnalogFilter(&hi2c1, I2C_ANALOGFILTER_ENABLE) !=  HAL_OK)
229     {
230       Error_Handler();
231     }
232 
233   }
234 
235   /* SPI1 init function */
236   void MX_SPI1_Init(void)
237   {
238 
239     hspi1.Instance = SPI1;
240     hspi1.Init.Mode = SPI_MODE_MASTER;
241     hspi1.Init.Direction = SPI_DIRECTION_2LINES;
242     hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
243     hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
244     hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
245     hspi1.Init.NSS = SPI_NSS_SOFT;
246     hspi1.Init.BaudRatePrescaler =  SPI_BAUDRATEPRESCALER_16;
247     /* SPI baudrate is set to 5.6 MHz (PCLK2/SPI_BaudRatePrescaler = 90/16 = 5.625  MHz)
248         to verify these constraints:
249         ILI9341 LCD SPI interface max baudrate is 10MHz for write and 6.66MHz for  read
250         l3gd20 SPI interface max baudrate is 10MHz for  write/read
251     PCLK2 frequency is set to 90 MHz 252
        */
253     hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
254     hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
255     hspi1.Init.CRCCalculation =  SPI_CRCCALCULATION_DISABLE;
256     hspi1.Init.CRCPolynomial = 7;
257     hspi1.Init.CRCLength =  SPI_CRC_LENGTH_DATASIZE;
258     hspi1.Init.NSSPMode = SPI_NSS_PULSE_ENABLE;
259 
260     //INICIALIZAMOS PONES  SPI
261     HAL_SPI_MspInit(&hspi1);
262 
263     if (HAL_SPI_Init(&hspi1) != HAL_OK)
264     {
265       Error_Handler();
266     }
267 
268   }
269 
270   /* TIM4 init function */
271   static void MX_TIM4_Init(void)
272   {
273 
274     TIM_MasterConfigTypeDef  sMasterConfig;
275     TIM_OC_InitTypeDef sConfigOC;
276 
277     htim4.Instance = TIM4;
278     htim4.Init.Prescaler = 24;
279     htim4.Init.CounterMode =  TIM_COUNTERMODE_UP;
280     htim4.Init.Period = 255;
281     htim4.Init.ClockDivision =  TIM_CLOCKDIVISION_DIV1;
282     if (HAL_TIM_PWM_Init(&htim4) !=  HAL_OK)
283     {
284       Error_Handler();
285     }
286 
287     sMasterConfig.MasterOutputTrigger =  TIM_TRGO_RESET;
288     sMasterConfig.MasterSlaveMode =  TIM_MASTERSLAVEMODE_DISABLE;
289     if (HAL_TIMEx_MasterConfigSynchronization(&htim4, &sMasterConfig) !=  HAL_OK)
290     {
291       Error_Handler();
292     }
293 
294     sConfigOC.OCMode =  TIM_OCMODE_PWM1;
295     sConfigOC.Pulse = 0;
296     sConfigOC.OCPolarity =  TIM_OCPOLARITY_HIGH;
297     sConfigOC.OCFastMode =  TIM_OCFAST_DISABLE;
298     if (HAL_TIM_PWM_ConfigChannel(&htim4, &sConfigOC, TIM_CHANNEL_4) !=  HAL_OK)
299     {
300       Error_Handler();
301     }
302      if (HAL_TIM_PWM_ConfigChannel(&htim4, &sConfigOC, TIM_CHANNEL_3) !=  HAL_OK)
303     {
```

```
304       Error_Handler();
305     }
306
307     HAL_TIM_MspPostInit(&htim4);
308
309   }
310
311   /** Configure pins  as
312           * Analog
313           * Input
314           * Output
315           * EVENT_OUT
316           * EXTI
317        PA11   ------> USB_DM
318        PA12   ------> USB_DP
319   */
320   static void MX_GPIO_Init(void)
321   {
322
323     GPIO_InitTypeDef  GPIO_InitStruct;
324
325     /* GPIO Ports Clock Enable  */
326     __HAL_RCC_GPIOE_CLK_ENABLE();
327     __HAL_RCC_GPIOC_CLK_ENABLE();
328     __HAL_RCC_GPIOF_CLK_ENABLE();
329     __HAL_RCC_GPIOA_CLK_ENABLE();
330     __HAL_RCC_GPIOD_CLK_ENABLE();
331     __HAL_RCC_GPIOB_CLK_ENABLE();
332
333     /*Configure GPIO pins : DRDY_Pin MEMS_INT3_Pin MEMS_INT4_Pin  MEMS_INT1_Pin
334                              MEMS_INT2_Pin  */
335     GPIO_InitStruct.Pin  = DRDY_Pin|MEMS_INT3_Pin|MEMS_INT4_Pin|MEMS_INT1_Pin
336                            |MEMS_INT2_Pin;
337     GPIO_InitStruct.Mode =  GPIO_MODE_EVT_RISING;
338     GPIO_InitStruct.Pull =  GPIO_NOPULL;
339     HAL_GPIO_Init(GPIOE, &GPIO_InitStruct);
340
341     /*Configure GPIO pins : CS_I2C_SPI_Pin LD4_Pin LD3_Pin  LD5_Pin
342                              LD7_Pin LD9_Pin LD10_Pin  LD8_Pin
343                              LD6_Pin  */
344     GPIO_InitStruct.Pin =  CS_I2C_SPI_Pin|LD4_Pin|LD3_Pin|LD5_Pin
345                            |LD7_Pin|LD9_Pin|LD10_Pin|LD8_Pin
346                            |LD6_Pin;
347     GPIO_InitStruct.Mode =  GPIO_MODE_OUTPUT_PP;
348     GPIO_InitStruct.Pull =  GPIO_NOPULL;
349     GPIO_InitStruct.Speed =  GPIO_SPEED_FREQ_LOW;
350     HAL_GPIO_Init(GPIOE, &GPIO_InitStruct);
351
352
353
354   //INICIALIZAO PIN DEL BOTON  DE USUARIO
355
356     GPIO_InitStruct.Pin =  GPIO_PIN_0;
357     GPIO_InitStruct.Mode =  GPIO_MODE_INPUT;
358     GPIO_InitStruct.Pull =  GPIO_NOPULL;
359     HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
360
361
362
363     /*Configure GPIO pin Output Level  */
364     HAL_GPIO_WritePin(GPIOE,  CS_I2C_SPI_Pin|LD4_Pin|LD3_Pin|LD5_Pin
365                            |LD7_Pin|LD9_Pin|LD10_Pin|LD8_Pin
366                            |LD6_Pin,  GPIO_PIN_RESET);
367
368    /*Configure GPIO pins :  P8 PB9   utilizados para activar los  motores*/
369     GPIO_InitStruct.Pin =  GPIO_PIN_8|GPIO_PIN_9;
370     GPIO_InitStruct.Mode =  GPIO_MODE_OUTPUT_PP;
371     GPIO_InitStruct.Pull =  GPIO_NOPULL;
372     GPIO_InitStruct.Speed =  GPIO_SPEED_FREQ_LOW;
373     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
374
375     HAL_GPIO_WritePin(GPIOB,  GPIO_PIN_8|GPIO_PIN_9,GPIO_PIN_RESET);
376
377
378     /*Configure GPIO pin : PB5 Utilizado para activar el Driver  */
379     GPIO_InitStruct.Pin =  GPIO_PIN_5;
```

```
380     GPIO_InitStruct.Mode =  GPIO_MODE_OUTPUT_PP;
381     GPIO_InitStruct.Pull =  GPIO_NOPULL;
382     GPIO_InitStruct.Speed =  GPIO_SPEED_FREQ_LOW;
383     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
384
385     HAL_GPIO_WritePin(GPIOB,  GPIO_PIN_5,  GPIO_PIN_RESET);
386
387   }
388
389   /* USER CODE BEGIN  4 */
390
391   /* USER CODE END  4 */
392
393   /**
394     * @brief  This function is executed in case of error  occurrence.
395     * @param  None
396     * @retval None
397     */
398   void  Error_Handler(void)
399   {
400     /* USER CODE BEGIN Error_Handler  */
401     /* User can add his own implementation to report the HAL error return state  */
402     while(1)
403     {
404     }
405     /* USER CODE END Error_Handler  */
406   }
407
408   #ifdef  USE_FULL_ASSERT
409
410   /**
411      * @brief Reports the name of the source file and the source line  number
412      * where the assert_param error has  occurred.
413      * @param file: pointer to the source file  name
414      * @param line: assert_param error line source  number
415      * @retval None
416      */
417   void assert_failed(uint8_t* file, uint32_t  line)
418   {
419     /* USER CODE BEGIN 6  */
420     /* User can add his own implementation to report the file name and line  number,
421       ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line)  */
422     /* USER CODE END  6 */
423
424   }
425
426   #endif
427
428   /**
429     * @}
430     */
431
432   /**
433     * @}
434   */
435
436   /************************ (C) COPYRIGHT STMicroelectronics *****END  OF FILE****/
437
```

11.1.2. Código fichero motores.c

```
1
2   /*************************MOTORES.C*************************
3   Nombre fichero: MOTORES.C
4   Descripción: En este fichero se implementarán todas las funciones propias que necesitamos  para
5   trabajar con el motor.
6
7   ***********************************************************************/
8
9   #include "stm32f3xx_hal.h"
10  #include "math.h"
11
12  #include "motores.h"
13
14  TIM_HandleTypeDef htim4;
15
16   int sentido_giro;
17
18  void moverMotor(int );
19  void paraMotor(void);
20  void subirMotor(int );
21  void bajarMotor(int );
22
23  int leerBoton(void);
24  void Leer_Aceleracion_XYZ(int16_t *);
25  uint16_t Leer_Aceleracion_X(int16_t pData);
26  void calculoAnguloY(double *);
27  void calculoAngulos(double *, double *);
28
29  /**********************moverMotor
30  * @brief   Función que mueve el motor según el sentido, horario o antihorario, podemos  seleccionar
31  diferentes velocidades mediante el sistema PWM
32
33    * @param  sentido :   indica el sentido 1--> horario 2--> antihorario
34    * @retval Nada
35
36  */
37  void moverMotor(int sentido)
38  {
39  int velocidad;
40  velocidad= (sentido==2) ? VELOCIDAD_SUBIDA  : VELOCIDAD_BAJADA;     // definimos la velocidad según
    suba o baje
41
42      sentido_giro=sentido;   // guardamos sentido
43    // ponemos el sentido
44     switch(sentido){
45       case 1: // bajando
46           __HAL_TIM_SetCompare(&htim4,TIM_CHANNEL_3,255); // damo un  impulso
47          HAL_Delay(10);
48          __HAL_TIM_SetCompare(&htim4,TIM_CHANNEL_4,0);
49          HAL_Delay(10);
50          __HAL_TIM_SetCompare(&htim4,TIM_CHANNEL_3,velocidad);   // activamos a la velocidad
51          HAL_Delay(10);
52          __HAL_TIM_SetCompare(&htim4,TIM_CHANNEL_4,0);
53          HAL_Delay(10);
54            break;
55       case 2: //subiendo
56         __HAL_TIM_SetCompare(&htim4,TIM_CHANNEL_4,255);    // damos un impulso
57          HAL_Delay(10);
58          __HAL_TIM_SetCompare(&htim4,TIM_CHANNEL_3,0);
59          HAL_Delay(10);
60          __HAL_TIM_SetCompare(&htim4,TIM_CHANNEL_4,velocidad);   // activamos a la velocidad
61          HAL_Delay(10);
62          __HAL_TIM_SetCompare(&htim4,TIM_CHANNEL_3,0);
63          HAL_Delay(10);
64            break;
65      }
66   }   // final de moverMotor
67
68   /**********************paraMotor
69  * @brief   para el motor poniendo todos los PWM  a cero
70
71    * @param  Nada
72    * @retval Nada
73
74  */
```

```
 void paraMotor()
 {

          __HAL_TIM_SetCompare(&htim4,TIM_CHANNEL_3,0);
 __HAL_TIM_SetCompare(&htim4,TIM_CHANNEL_4,0); 81   }

 /**********************subirMotor
* @brief   Función auxiliar que sube el motor durante un tiempo determinado, tras el cual para  los motores
   * @param  Nada
  * @retval Nada

*/
 void subirMotor(int t)
 {
   moverMotor(2);
   HAL_Delay(t);
   paraMotor();
 HAL_Delay(1000);
 }

 /**********************bajarMotor
  * @brief   Función auxiliar que baja el motor durante un tiempo determinado,  tras el cual para los
motores
  * @param  Nada
  * @retval Nada

*/

 void bajarMotor(int t)
 {
   moverMotor(1);
   HAL_Delay(t);
   paraMotor();
   HAL_Delay(1000);
 }

 /**********************leerBoton
* @brief     Nos devuelve el estado del pin PA0, ( nos devuelve si el boton del  usuario está pulsado o no)
 devuelve 0 si no esta pulsado, 1 si el boon está  pulsado
  * @param  Nada
* @retval el estado del boton usuario (0-->  no pulsado     1--> pulsado)
*/
 int leerBoton(void)
{
  int estadoBotonPA0;
  estadoBotonPA0=HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_0);
  HAL_Delay(10);  //evita rebotes.

  return estadoBotonPA0;
}

/**********************calculoAngulos
* @brief     Devuelve los alngulos pitch y row en base a la aceleración aleida,  devuelve los angulos en grados
  * @param  *p, *r:punteros en los  que se almacenaran los angulos pitch y  row,
* @retval nada
*/



void calculoAngulos(double *p, double  *r){
int16_t pData[3];

  Leer_Aceleracion_XYZ(pData); //  leemos aceleración

  *r=asin(pData[1]/1000.0);   // pasamos de aceleración  a angulos
  *p=asin(pData[0]/1000.0);

  (*p)=(*p)*180/PI;      //pasamos a grados
  (*r)=(*r)*180/PI;
```

```
149   }
150
151   /**********************calculoAnguloY
152     * @brief   Devuelve los alngulos (angulo girado alrededor del eje Y - cabeceo- )  pitch
153   en base a la aceleración leida sobre el eje X, devuelve el angulo en  grados
154     * @param  *p:puntero en el  que se almacenara el angulo  pitch
155   * @retval nada
156   */
157
158   void calculoAnguloY(double  *p){
159     int16_t pData;
160
161     pData=Leer_Aceleracion_X(pData); // leemos aceleración a lo largo del  eje X.
162
163
164
165
166               *p=asin(pData/1000.0);   // calculamos el ángulo
167
168
169     //pasamos a grados
170
171          (*p)=(*p)*180/PI;
172
173
174
175   }
176
177
```

11.1.3. Código fichero medidas.c

```
1
2   /*****************************MEDIDAS.C*****************************
3   Nombre fichero: MEDIDAS.C
4   Descripción: En este fichero se implementarán todas las  funciones
5   de alto nivel , que serán llamadas desde la función principal.
6
7   ***********************************************************************/
8
9
10  #include "stm32f3xx_hal.h"
11  #include "math.h"
12
13  #include "motores.h"
14
15  #define WS   2.285    //velocidad angular subida medidas  aparte
16  #define WB  1.5     //velocidad angular bajada
17
18
19  //definicion de funciones
20
21  void calculoAngulos(double *, double  *);
22  void calculoAnguloY(double  *);
23  void moverMotor(int);
24  void paraMotor(void);
25  int leerBoton(void);
26  void inicializacionKalmanAcelerometro(double ,double  );
27  double filtroKalmanAcelerometro(double  );
28  void inicializacionKalmanGiroscopo(double  );
29  double filtroKalmanGiroscopo(double  );
30  void inicializacionKalmanAcelerometroGiroscopo(double  );
31  double filtroKalmanAcelerometroGiroscopo(double ,double,  float);
32  void Leer_VelocidadAngular_XYZ(float  *);
33  int16_t Leer_Aceleracion_X(int16_t  pData);
34  double CalculoAnguloGiroscopo(double  );
35  HAL_StatusTypeDef  InicializarAcelerometro(void);
36  HAL_StatusTypeDef InicializarGiroscopo(void);
37
38  // funciones implementadas en  el fichero
39
40  void  Nivelar_Acelerometro(void);
41  void  Nivelar_Giroscopo(void);
42  void  Nivelar_Giroscopo_Acelerometro(void);
43  void  AnguloAcelerometro(void);
44  void  AnguloGiroscopo(void);
45  void Angulo_Giroscopo_Acelerometro(void);
46
47  double pasarKalmanAcelerometro(double  );
48  void InicializarSensores(void);
49
50  extern long int tiempo1;
51
52  /***************  InicializarSensores
53  **
54    * @brief  Inicializa tanto el acelerómetro como el giróscopo, se limita a llamar la  las
55    funciones de inicialización implementadas en sus respectivos  ficheros
56    * @param  nada
57    * @retval nada
58    */
59
60  void  InicializarSensores(void){
61
62    printf("Empezamos Inicializacion  \n\r");
63        if  (InicializarAcelerometro()!=HAL_OK){
64                printf("\n Error en Inicializacion  Acelerometro");
65                while(1){}
66                }
67        if  (InicializarGiroscopo()!=HAL_OK){
68                 printf("\n Error en Inicializacion  Giróscopo");
69                while(1){}
70              }
71  printf("\n Giróscopo y  Acelerómetro  inicializado");
72
73  }
```

```
74
75
76  /***************  Nivelar_Acelerometro*****************
77  **
78  * @brief    Función que nivela la plataforma dejandola con un desnivel entre [-1°,+1°],  tomando
79  solo los datos proporcionados por el acelerómetro. Se pasará el filtro de Kalman para el
    acelerómetro.
80  Se deberá tener en cuenta que para que las medidas sean fiables cuando se realicen las medidas la
81  plataforma debera estar quieta. (de lo contrario se estaría midiendo la aceleración centrípeta de
    la
82  plataforma por lo que nos daría valores  erroneos).
83    Como se ve se calcula el tiempo de subida o  bajada.
84    * @param  nada
85    * @retval nada
86    */
87
88
89  void Nivelar_Acelerometro(void){
90
91    double pitch,row;
92    double w,tiempo;
93    long int i=0;
94
95    calculoAngulos(&pitch, &row); // devuelve angulos  en grados
96
97
98    pitch=pasarKalmanAcelerometro(pitch);
99
100
101
102     while  (fabs(pitch)>0.4)
103   {
104     int giro;
105
106     giro=( pitch>0 ? 1 :2 );
107     w=(pitch>0 ? WB :WS); //si giro==1 tiene que bajar, si giro==2 tiene  que subir
108
109     tiempo=pitch/w; //tiempo de subida o  de bajada
110
111     printf("\rMedida Num , tiempo que deberá moverse%lf  ms \n",tiempo*1000);
112
113     moverMotor(giro);
114     HAL_Delay(tiempo*1000);
115     paraMotor();
116     HAL_Delay(1000);
117
118     i++;
119 calculoAngulos(&pitch,  &row);
120 printf("\rMedida Num,%ld antes filtro: .- Angulo Pitch (giro alrededor del eje Y-cabeceo):%lf
    (grados)\n",i,pitch);
121
122     pitch=pasarKalmanAcelerometro(pitch);
123
124     printf("\rMedida Num,%ld despues de filtro: .- Angulo îtch (giro alrededor  del eje
    Y-cabeceo):%lf  (grados)\n",i,pitch);
125
126     //HAL_Delay(50);
127   }
128
129 }
130
131 /***************  Nivelar_Giroscopo*****************
132 **
133 * @brief    Función que nivela la plataforma dejandola con un desnivel entre [-1°,+1°],  tomando
134 solo los datos proporcionados por el giróscopo. Se pasará el filtro de Kalman para el  giróscopo.
135 Se deberá tener en cuenta que para que las medidas sean fiables la velocidad angular se supone
    constante
136 para que siga el modelo descrito en el calculo del angulo medante el  giróscopo.
137 angulo=angulo_anterior+w*dt
138 Para inicializar el angulo se utiliza el acelerómetro.(plataforma estática)
139
140   * @param  nada
141   * @retval nada
142   */
143
144 void Nivelar_Giroscopo(void){
145
```

```
146     double pitch,row;
147     double w;
148     long int i=0;
149     int giro,giro_ant;
150
151
152   calculoAngulos(&pitch, &row); // devuelve angulos en grados por el acelerometro, para el angulo
      inicial
153     pitch=pasarKalmanAcelerometro(pitch);
154
155   if (fabs(pitch)>1){
156     giro=( pitch>0 ? 1 :2 );
157     w=(pitch>0 ? WB :WS);
158     inicializacionKalmanGiroscopo(w); // inicializamos velocidad angular con la medida  off-line
159     giro_ant=giro;
160     tiempo1=0;  //inicializamos intérvalo de  tiempo
161      while (fabs(pitch)>1)
162     {
163
164       giro=( pitch>0 ? 1 :2 );
165   w=(pitch>0 ? WB :WS);
166       if(giro!=giro_ant){   //cambiamos de sentido de  giro
167            paraMotor();
168   inicializacionKalmanGiroscopo(w);            // inicializamos velocidad angular con  la medida
      off-line
169            giro_ant=giro;
170            tiempo1=0;  //inicializamos  tiempo
171            HAL_Delay(100);
172       }
173
174       if( !tiempo1)
175       {
176         tiempo1=HAL_GetTick();   // empezamos a contar el intérvalo de  tiempo.
177
178       moverMotor(giro);      // movemos el motor
179       }
180
181         pitch=CalculoAnguloGiroscopo(pitch); //calculamos el angulo mediante el  giróscopo, segun
      modelo
182       i++;
183       printf("\n\rAngulo ,medida %ld:%lf  (grados)\n",i,pitch);
184     }
185       paraMotor();
186
187   }
188     printf("\n\r FIN DE LAS MEDIDAS,  nivelado");
189
190
191   }
192
193   /***************  Angulo_Giroscopo_Acelerometro*****************
194   **
195     * @brief  Función que nos mide el ángulo en cada momento,  fusionando
196   los datos entregado por giróscopo y por acelerómetro mediante el filtro de kalman  desarrollado  a
      tal
197   efecto, Como ángulo inicial se toma el proporcionado par el acelerómetro( estando la  plataforma
      estática).
198   Y el modelo lineal que se sigue es angulo=angulo_anterior+w*dt, donde el ángulo como la  deriva
      del giróscopo son
199   estimados por el filtro,
200     A fin de que las medidas del acelerómetro sean lo más exactas posibles es deseable que  la
      velocidad
201   angular sea lo más baja  posible.
202     Tener en cuenta que la plataforma pueda subir o bajar durante el tiempo  estimado
203
204     * @param  nada
205     * @retval nada
206     */
207   void  Angulo_Giroscopo_Acelerometro(void){
208     double pitch;
209     double u,dt,t2;
210     float pfData[3]={0};
211     double tiempo;
212     long int i;
213
214
```

```
215     calculoAnguloY(&pitch);   // calculo ángulo inicial
216     printf("\n Antes del filtro:  %lf",pitch);
217     pitch=pasarKalmanAcelerometro(pitch);
218       printf("\n\rAngulo Inicial:%lf (grados)\n",pitch);
219
220     while(!leerBoton()){}   // esperamos mientras no pulsamos boton de  seguir
221       HAL_Delay(1000);
222
223     //tiempo1=HAL_GetTick();
224     tiempo=HAL_GetTick();    // inicialiozamos tiempo de subir  o bajar
225
226     moverMotor(1); //2->subimos motor 2 segundos, transcurridos dos segundos parará 1->  bajamos
227     // HAL_Delay(100);
228       tiempo1=HAL_GetTick();   // inicializamos tiempo del  primer intérvalo
229         i=0;
230
231     Leer_VelocidadAngular_XYZ(pfData);   //calculamos velocidad angular girósopo
232
233     inicializacionKalmanAcelerometroGiroscopo(pitch);   // inicializamos filtro  de kalman
234
235
236     while((HAL_GetTick()-tiempo)<4000){ //vamos calculando durante  4 segundos
237       i++;
238
239       calculoAnguloY(&pitch);   // calculamos  angulo acelerómetro
240
241
242
243       Leer_VelocidadAngular_XYZ(pfData);   // calculamos velocidad angular  giróscopo
244       u=pfData[0];
245       t2=HAL_GetTick();   // medimos final intérvalo  tiempo
246       dt=t2-tiempo1;     // calculamos intérvalo  tiempo
247       tiempo1=t2;        // inicializamos siguiente intérvalo  tiempo
248       printf("\n\rAngulo antes del filtro,medida %ld:%lf  (grados)\n",i,pitch);
249       pitch=filtroKalmanAcelerometroGiroscopo( pitch,  dt,  u);   // calculamos ángulo  mediante  filtro
250
251       printf("\n\rAngulo despues del filtro ,medida  %ld:%lf  (grados)\n",i,pitch);
252
253     }
254     paraMotor();   // finalizado el tiempo  paramos.
255     printf("\n\r FIN DE LAS MEDIDAS");
256
257   }
258
259   /***************  Nivelar_Giroscopo_Acelerometro*****************
260   **
261   * @brief    Función que nivela la plataforma dejandola con un desnivel entre [-1°,+1°],  fusionando
262   los datos entregado por giróscopo y por acelerómetro mediante el filtro de kalman desarrollado a
      tal
263   efecto, Como ángulo inicial se toma el proporcionado par el acelerómetro( plataforma estática). Y
      el modelo
264   lineal que se sigue es angulo=angulo_anterior+w*dt, donde el ángulo como la deriva del giróscopo
      son
265   estimados por el  filtro,
266   A fin de que las medidas del acelerómetro sean lo más exactas posibles es deseable que la
      velocidad
267   angular sea lo más baja posible.
268
269     * @param  nada
270     * @retval nada
271     */
272
273   void  Nivelar_Giroscopo_Acelerometro(void){
274
275     double pitch;
276     double u,dt,t2;
277     long int i=0;
278     int giro,giro_ant;
279     float pfData[3]={0};
280
281     // angulo inicial
282
283
284     calculoAnguloY(&pitch); // devuelve angulos en grados por el  acelerometro
285     pitch=pasarKalmanAcelerometro(pitch);
286
```

```
287
288
289   if (fabs(pitch)>0.5){
290     giro=( pitch>0 ? 1 :2 );
291
292     inicializacionKalmanAcelerometroGiroscopo(pitch); // inicializamos el  filtro
293     giro_ant=giro;
294     tiempo1=0;
295      while (fabs(pitch)>0.5)
296     {
297
298       giro=( pitch>0 ? 1 :2 );
299
300       if(giro!=giro_ant){   //cambiamos de sentido de giro
301            paraMotor();
302            inicializacionKalmanAcelerometroGiroscopo(pitch);
303            giro_ant=giro;
304            tiempo1=0;
305            HAL_Delay(100);
306       }
307
308       if( !tiempo1)
309       {
310         tiempo1=HAL_GetTick();   // inicializamos intérvalo
311
312         moverMotor(giro);
313       }
314
315
316       calculoAnguloY(&pitch); // calculamos angulo mediante el  acelerómetro
317       printf("\n pich  %lf",pitch);
318       Leer_VelocidadAngular_XYZ(pfData);   //calculamos velocidad angular mediante  giróscopo
319       u=pfData[0];
320       printf("\n vel angular:  %lf",u);
321       t2=HAL_GetTick();  // fin del  intérvalo
322       dt=t2-tiempo1;                        // intérvalo de  tiempo
323       tiempo1=t2;  // inicializamos intérvalo de  giro
324       pitch=filtroKalmanAcelerometroGiroscopo( pitch,  dt,  u);  //fusión de datos mediante  le filtro
325       i++;
326       printf("\n\rAngulo ,medida  %ld:%lf  (grados)\n",i,pitch);
327
328     }
329       paraMotor();
330
331   }
332     printf("\n\r FIN DE LAS MEDIDAS, nivelado Acelerometro  - Giroscopo");
333
334
335   }
336
337   /***************  AnguloAcelerómetro*****************
338   **
339   * @brief    Función que calcula el ángulo que forma la plataforma(pitch)  tomando
340   solo los datos proporcionados por el acelerómetro. Se pasará el filtro de Kalman para el
      acelerómetro.
341   Se deberá tener en cuenta que para que las medidas sean fiables cuando se realicen las medidas la
342   plataforma debera estar  quieta.
343     Se compara la medida antes de pasar el filtro con la medida tras pasar el filtro de  kalman.
344     * @param  nada
345     * @retval nada
346     */
347
348
349   void AnguloAcelerometro(void){
350     double pitch,row;
351
352
353
354
355   calculoAnguloY(&pitch);
356   printf("\rMedida Num, antes filtro: .- Angulo Row (wx),Pitch (wy):%lf,%lf
      (grados)\n",row,pitch);
357
358     pitch=pasarKalmanAcelerometro(pitch);     // se realizan 5 medidas a fin de estimar el valor  cte.
359                                               // del ángulo
360
```

```
362       printf("\rMedida Num,despues de filtro:Pitch (giro alr. del  eje Y-cabeceo):%lf
    (grados)\n",pitch);
363
364   }
365
366   /***************  AnguloGiróscopo*****************
367   **
368     * @brief  Nos da el ángulo medido medante el algoritmo del giróscopo, cuando subimos o  bajamos
    la plataforma durante un tiempo, en la ventana del debugger nos aparecer´á los angulos medidos  en
    cada instante, y por lo tanto .el  final.
369   El ángulo inicial se toma el dado por el aceleróemtro, (plataforma  estática).
370   ASEGURARSE QUE LA PLATALFORMA PODRÁ SUBIR o BAJAR DURANTE dicho  tiempo
371     Se compara la medida antes de pasar el filtro con la medida tras pasar el filtro de  kalman.
372     * @param  nada
373     * @retval nada
374     */
375
376
377   void AnguloGiroscopo(){
378     long int i,tiempo;
379     double row,pitch;
380     float pfData[3]={0.0};
381
382
383     calculoAngulos(&pitch, &row); // calculamos angulo inicial con  el acelerómetro
384
385     pitch=pasarKalmanAcelerometro(pitch);
386       printf("\n\rAngulo Inicial:%lf (grados)\n",pitch);
387
388
389     tiempo1=HAL_GetTick();   // inicializamos tiempo del  intérvalo
390     tiempo=tiempo1;
391
392     moverMotor(2); //subimos motor
393
394
395         i=0;
396
397     Leer_VelocidadAngular_XYZ(pfData);   // leemos velocidad angular
398
399     inicializacionKalmanGiroscopo(pfData[0]); // inicializamos kalman con  dicho valor
400
401     while((HAL_GetTick()-tiempo)<2000){ //vamos enseñando los angulo calculados mediante  el giróscop
402       i++;
403       pitch=CalculoAnguloGiroscopo(pitch);
404       printf("\n\rAngulo ,medida  %ld:%lf  (grados)\n",i,pitch);
405
406     }
407     paraMotor();
408     printf("\n\r FIN DE  LAS MEDIDAS");
409
410
411
412
413
414
415   }
416
417   /***************  pasarKalmanAcelerómetro*****************
418   **
419     * @brief  Función auxiliar al filtro de kalman, para poder estimar una valor  constante
    mediante el FK , se toman una serie de lecturas y el valor estimado converge el valor  real.
420     Se han tomado tan solo 5 medidas, se debería hacer un estudio  de convergencia.
421
422     * @param  angulo con el que se inicializa el filtro de kalman , para  estimar una magnitud
    constante
423     * @retval angulo estimado tras  el filtro.
424     */
425
426   double pasarKalmanAcelerometro(double  angulo)
427    {
428      double pitchF,pitch;
429      pitch=angulo;
430      // pasamos por el filtro de kalman , con 5 medidas suponemos que con esa cantidad  de medidas
    convergerá
431         inicializacionKalmanAcelerometro(pitch,1.00);
```

```
434         for(int j=1;j<5;j++){
435 
436           // volvemos a leer nuevas  entradas
437       //    calculoAngulos(&pitch,  &r);
438       calculoAnguloY(&pitch);
439 
440 
      pitchF=filtroKalmanAcelerometro(pitch);

441         }
442         return pitchF;
443 
444     }
445   // final de fichero
446 
```

11.1.4. Código fichero filtroKalman.c

```
/*****************************filtroKalman.c*************************
Nombre fichero: filtroKalman.C
Descripción: En este fichero se implementarán todas las funciones propias del filtro de Kalman, para
el acelerómetro, para el giróscopo y para el acelerómetro+giróscopo.


*********************************************************************/


#include "stm32f3xx_hal.h"

//variables utilizadas en el filtro de kalman, unidimensional (solo  acelerómetro).
double  x,xminus,xminus_1,x0;    //vetores de estado en difierentes  momentos
double  p,pminus,pminus_1,p0;     // matriz de covarianza del error en diferentes  momentos
double K;                       // gananzia de  kalmana
double Q;                       // matriz de covarianza del error del  modelo
double R;                       // matriz de covarianza del error de  la medida

//variables utilizadas en el filtro de kalman bidimensinal  (giróscopo ).
/*
variable de estado:
    x[0]---> velocidad angular suponemos que es constante  x[0]=xminus_1[0]
    x[1]---> ruido del giróscopo, aleatorio, suponemos que tambien constante x[1]=xminus_1[1]

    solo devolvemos x[0] la velocidad  angular estimada.

    Se devería llamar con x[0] con la velocidad angular estimada  manualmente
                          x[1] lo llamariamos con cero, (suponemos error inicial cero)

*/
double  xg[2],xminusg[2],xminus_1g[2],x0g[2];    //vetores de estado en difierentes  momentos
double  pg[2][2],pminusg[2][2],pminus_1g[2][2],p0g[2][2];     // matriz de covarianza del  error en diferentes  momentos
double Kg[2];                       // gananzia de  kalmana
double Qg[2][2];                        // matriz de covarianza del error del  modelo
double Rg;                        // matriz de covarianza del error de  la medida


void inicializacionKalmanGiroscopo(double  );
double filtroKalmanGiroscopo(double );

void inicializacionKalmanAcelerometro(double ,double  );
double filtroKalmanAcelerometro(double );


/*****************************FILTRO  KALMAN ACELEROMETRO*****************************
*************************************************************************************
*************************************************************************************/

/***************  InicializarKalmanAcelerometro
**
  * @brief  Inicializa el filtro de kalman, para el  acelerómetro.
  * @param  x0:angulo inicial,
  * @param  p0:matriz de covarianza inicial, normalmente es la  unidad
  * @retval nada
  */


void inicializacionKalmanAcelerometro(double  x0,double p0)
{
  xminus_1=x0;
  pminus_1=p0;
  // inicializamos las matrices de las covarianzas de los errores, es lo que  deberíamos
  //variar para ver la respuesta del filtro

  Q=0.0001;  //valores extraidos de  la bibliografia
  R=0.07;


}

/***************  filtroKalmanAcelerometro
**
  * @brief  Pasa el filro de kalman a una medida del  acelerómetro
```

```
74    * @param  z: medida proporcionada por el acelerometro. (realmente se trabaja con el  ángulo)
75    * @retval medida del ángulo estimada por el filtro
76    */
77  double filtroKalmanAcelerometro(double z)
78  {
79
80    // etapa de prediccion
81
82    xminus=xminus_1;
83    pminus=pminus_1+Q;
84
85    // etapa de correccion
86
87    K=pminus/(pminus+R);     //calculamos ganacia de  kalman.
88    x=xminus+K*(z-xminus);   // recalculamos el  estado
89    p=(1-K)*pminus;          // recalculamos la matriz de covarianza  del error.
90
91    // actualizamos xminus y pminus para próxima  interacción
92    xminus_1=x;
93    pminus_1=p;
94
95    return x;
96
97  }
98
99  /***************************FILTRO  KALMAN GIROSCOPO*****************************
100 ********************************************************************************
101 ********************************************************************************/
102
103 /***************  inicializacionKalmanGiroscopo
104 **
105   * @brief  funcion que inicializa los parametros fundamentales que utilizaremos en el  filtro
106 la velocidad angular inicial se pasa como parametro,  x0
107 el ruido inicial se toma como el bias medido aparte en nuestro caso  -0.1
108 la matriz de covarancia inicial se toma la matriz unida [1 0;0  1]
109 los valores de Q y R se toman de la  bibliografía
110   * @param  x0g:velocidad angular para inicializar el filtro se toma como -0.1 medido  aparte
111   * @retval nada
112   */
113
114
115 void inicializacionKalmanGiroscopo(double  x0g)
116 {
117   // la velocidad angular inicializamos al parametro pasado, el ruido inicial  a cero
118   xminus_1g[0]=x0;xminus_1g[1]=-0.1;
119   // la matriz  de covarianza  inicial se toma como la  matriz identidad.
120   pminus_1g[0][0]=1;pminus_1g[0][1]=0;pminus_1g[1][0]=0;pminus_1g[1][1]=1;
121
122   // inicializamos las matrices de las covarianzas de los errores, es lo que  deberíamos
123   //variar para ver la respuesta del filtro
124
125   Qg[0][0]=0.001;Qg[0][1]=0;Qg[1][0]=0;Qg[1][1]=0.003;   //valores extraidos de  la bibliografia
126   Rg=0.07;
127
128
129 }
130
131 /***************  filtroKalmanGiróscopo
132 **
133   * @brief  Pasa el filro de kalman a una medida del  giróscopo
134   * @param  z: medida proporcionada por el giróscopo. (realmente se trabaja con velocidad  angular)
135   * @retval devuelve la velocidad angular estimada - deriva estimada.
136   */
137
138
139 double filtroKalmanGiroscopo(double  z)
140 {
141
142   // etapa de prediccion  U=Uanterior -deriva
143
144   xminusg[0]=xminus_1g[0]-xminus_1g[1];xminusg[1]=xminus_1g[1];
145
146   pminusg[0][0]=pminus_1g[0][0]-pminus_1g[1][0]-pminus_1g[0][1]+pminus_1g[1][1]+Qg[0][0];
147   pminusg[0][1]=pminus_1g[0][1]-pminus_1g[1][1]+Qg[0][1];
148   pminusg[1][0]=pminus_1g[1][0]-pminus_1g[1][1]+Qg[1][0];
149   pminusg[1][1]=pminus_1g[1][1]+Qg[1][1];
150
```

```
151     // etapa de correccion
152
153     Kg[0]=pminusg[0][0]/(pminusg[0][0]+Rg);    //calculamos ganacia de kalman.
154     Kg[1]=pminusg[1][0]/(pminusg[0][0]+Rg);
155
156     xg[0]=xminusg[0]+Kg[0]*(z-xminusg[0]);   // recalculamos el estado
157     xg[1]=xminusg[1]+Kg[1]*(z-xminusg[0]);
158
159       // recalculamos la matriz de covarianza del error.
160
161     pg[0][0]=(1-Kg[0])*pminusg[0][0];  pg[0][1]=(1-Kg[0])*pminusg[0][1];
162     pg[1][0]=pminusg[1][0]-Kg[1]*pminusg[0][0];   pg[1][1]=pminusg[1][1]-Kg[1]*pminusg[0][1];
163
164
165     // actualizamos xminus y pminus para próxima interacción
166     xminus_1g[0]=xg[0];xminus_1g[1]=xg[1];
167     pminus_1g[0][0]=pg[0][0];  pminus_1g[0][1]=pg[0][1];
168     pminus_1g[1][0]=pg[1][0];  pminus_1g[1][1]=pg[1][1];
169
170
171     return (xg[0]-xg[1]);
172
173   }
174
175   /*************************FILTRO  KALMAN GIROSCOPO+ACELEROMETRO*****************************
176   ***************************************************************************************
177   ***************************************************************************************/
178
179
180   /***************  inicializacionKalmanGiroscopo
181   **
182     * @brief  funcion que inicializa los parametros fundamentales que utilizaremos en el  filtro
183   // El angulo inicial se pasa como parametro,  x0
184   //la deriva inicial ponemos la calculada aparte (media de 600 medidas) la tomamos como  -0.1
185   // la matriz de covarancia inicial se toma la matriz unida [1 0;0  1]
186   // los valores de Q y R se toman de la  bibliografía
187     * @param  x0g:angulo inicial para inicializar el  filtro
188     * @retval nada
189     */
190
191
192
193   void inicializacionKalmanAcelerometroGiroscopo(double  x0g)
194   {
195     // la angulo inicial la deriva inicial ponemos la calculada aparte (media de  600 medidas)
196     xminus_1g[0]=x0g;xminus_1g[1]=-0.1;
197     // la matriz de covarianza  inicial se toma como la  matriz identidad.
198     pminus_1g[0][0]=1;pminus_1g[0][1]=0;pminus_1g[1][0]=0;pminus_1g[1][1]=1;
199
200     // inicializamos las matrices de las covarianzas de los errores, es lo que  deberíamos
201     //variar para ver la respuesta del filtro
202
203     Qg[0][0]=0.001;Qg[0][1]=0;Qg[1][0]=0;Qg[1][1]=0.003;   //valores extraidos de  la bibliografia
204     Rg=0.05;
205
206
207   }
208
209   /***************  filtroKalmanAcelerometroGiroscopo
210   **
211     * @brief  Pasa el filro de kalman a una entrada del giróscopo  y medida  del acelerómetro  (
212   algoritmo explicado en detalle en el trabajo  correspondiente)
213     * @param  z: ángulo proporcionado por la medida del  acelerómetro.
214     * @param  dt: intérvalo de tiempo  medido
215     * @param  u: medida del giróscopo
216
217     * @retval medida estimada por el filtro  del ángulo.
218     */
219
220
221   double filtroKalmanAcelerometroGiroscopo(double z,double dt,float  u)
222   {
223
224     // etapa de prediccion  angulo=angulo_anterior+(u-deriva)*dt
225
226     dt/=1000; //dt en milisegundos, u en °/s, pasamos todo a  segundos
```

```
227
228     xminusg[0]=xminus_1g[0]-xminus_1g[1]*dt+u*dt;    //modelo del sistema
229     xminusg[1]=xminus_1g[1];
230
231
    pminusg[0][0]=pminus_1g[0][0]-pminus_1g[1][0]*dt-pminus_1g[0][1]*dt+pminus_1g[1][1]*dt*dt+Qg[0][0];
232     pminusg[0][1]=pminus_1g[0][1]-pminus_1g[1][1]*dt+Qg[0][1];
233     pminusg[1][0]=pminus_1g[1][0]-pminus_1g[1][1]*dt+Qg[1][0];
234     pminusg[1][1]=pminus_1g[1][1]+Qg[1][1];
235
236     // etapa de correccion
237
238     Kg[0]=pminusg[0][0]/(pminusg[0][0]+Rg);     //calculamos ganacia de kalman.
239     Kg[1]=pminusg[1][0]/(pminusg[0][0]+Rg);
240
241     xg[0]=xminusg[0]+Kg[0]*(z-xminusg[0]);   // recalculamos el estado
242     xg[1]=xminusg[1]+Kg[1]*(z-xminusg[0]);
243
244       // recalculamos la matriz de covarianza del error.
245
246     pg[0][0]=(1-Kg[0])*pminusg[0][0];  pg[0][1]=(1-Kg[0])*pminusg[0][1];
247     pg[1][0]=pminusg[1][0]-Kg[1]*pminusg[0][0];   pg[1][1]=pminusg[1][1]-Kg[1]*pminusg[0][1];
248
249
250     // actualizamos xminus y pminus para próxima interacción
251     xminus_1g[0]=xg[0];xminus_1g[1]=xg[1];
252     pminus_1g[0][0]=pg[0][0];  pminus_1g[0][1]=pg[0][1];
253     pminus_1g[1][0]=pg[1][0];  pminus_1g[1][1]=pg[1][1];
254
255
256     return xg[0];
257
258   }
259
```

11.2. Presupuesto

COMPONENTES ELECTRÓNICOS

MATERIALES				
Uds.	Descripción	Cantidad	Precio	Total
u	Tarjeta STM32F3	1	10.96 €	10.96 €
u	Driver para motor de DC	1	3.37€	3.37 €
u	Fuente de alimentación	1	27.73 €	27.73 €
u	Placa perforada de Prototipado	1	7.06 €	7.06 €
u	Tira de pines macho	1	0.13 €	0.13 €
u	Zócalo de 16 pines	1	0.24 €	0.24 €
u	Cables Hembra-Hembra para placas	8	0.27 €	2.22 €

Subtotal Materiales	51.71 €

MANO DE OBRA				
Uds.	Descripción	Cantidad	Precio	Total
h	Diseño del circuito	1	15.00 €	15.00 €
h	Soldadura y montaje	2	12.00 €	24.00 €

Subtotal Mano de obra	39.00 €

COSTE TOTAL DE FABRICACIÓN	90.71 €

COMPONENTES MECÁNICOS

MATERIALES				
Uds.	**Descripción**	**Cantidad**	**Precio**	**Total**
u	Kit brazo robótico	1	47.28 €	47.28 €
u	Lamina de contrachapado	1	4.65 €	4.65 €
u	TECHNIC BRICK 1X16, Ø4,9	4	1.05 €	4.20 €
u	TECHNIC 15M BEAM	1	0.31 €	0.31 €
u	CONNECTOR PEG W. FRICTION	2	0.01 €	0.02 €
u	Angular beam 90degr. w.4 snaps	2	0.32 €	0.64 €
u	TECHNIC 11M BEAM	2	0.10 €	0.20 €
u	1/2 BUSH	3	0.01 €	0.01 €
u	CROSS AXLE 8M	1	0.01 €	0.01 €
u	2X1X3 STEERING KNUCKLE ARM	1	0.04 €	0.04 €
u	MakeBlock Plate 7 x 9 B	1	3.54 €	3.54 €
u	Tornillo m4x20	2	0.11 €	0.22 €
u	Tornillo m5x30	2	0.33 €	0.66 €

Subtotal Materiales	61.78 €

MANO DE OBRA				
Uds.	Descripción	Cantidad	Precio	Total
h	Montaje del prototipo	2	15.00 €	30.00 €
h	Cortado y lijado contrachapado	0.5	15.00 €	7.50 €

Subtotal Mano de obra	37.50 €

COSTE TOTAL DE FABRICACIÓN	99.28 €

ESTUDIO, DESARROLLO Y PREPARACIÓN

MANO DE OBRA				
Uds.	**Descripción**	**Cantidad**	**Precio**	**Total**
h	Documentación previa y estudio	33.5	15.00 €	502.5 €
h	Programación código	150	15.00 €	2250.00 €
h	Pruebas finales del prototipo y Software	50	15.00 €	750.00 €
h	Redacción técnica del trabajo	50	5.00 €	250.00 €
h	Impresión y encuadernación	1	5.00 €	5.00 €
h	Preparación de la presentación final	10	5.00 €	50.00 €

Subtotal mano de obra	3807.50 €

COSTE TOTAL DEL DESARROLLO DEL TRABAJO	3997.49 €
TIEMPO TOTAL INVERTIDO	300 horas

RESUMEN DEL PRESUPUESTO DEL PROYECTO

Denominación	Materiales	Mano de obra	Total
Componentes electrónicos	51.71 €	39.00 €	90.71 €
Componentes mecánicos	61.78 €	37.50 €	99.28 €
Estudio, desarrollo y preparación	0.00 €	3807.50 €	3807.50 €

TOTAL	113.49 €	3884.00 €	3997.49 €

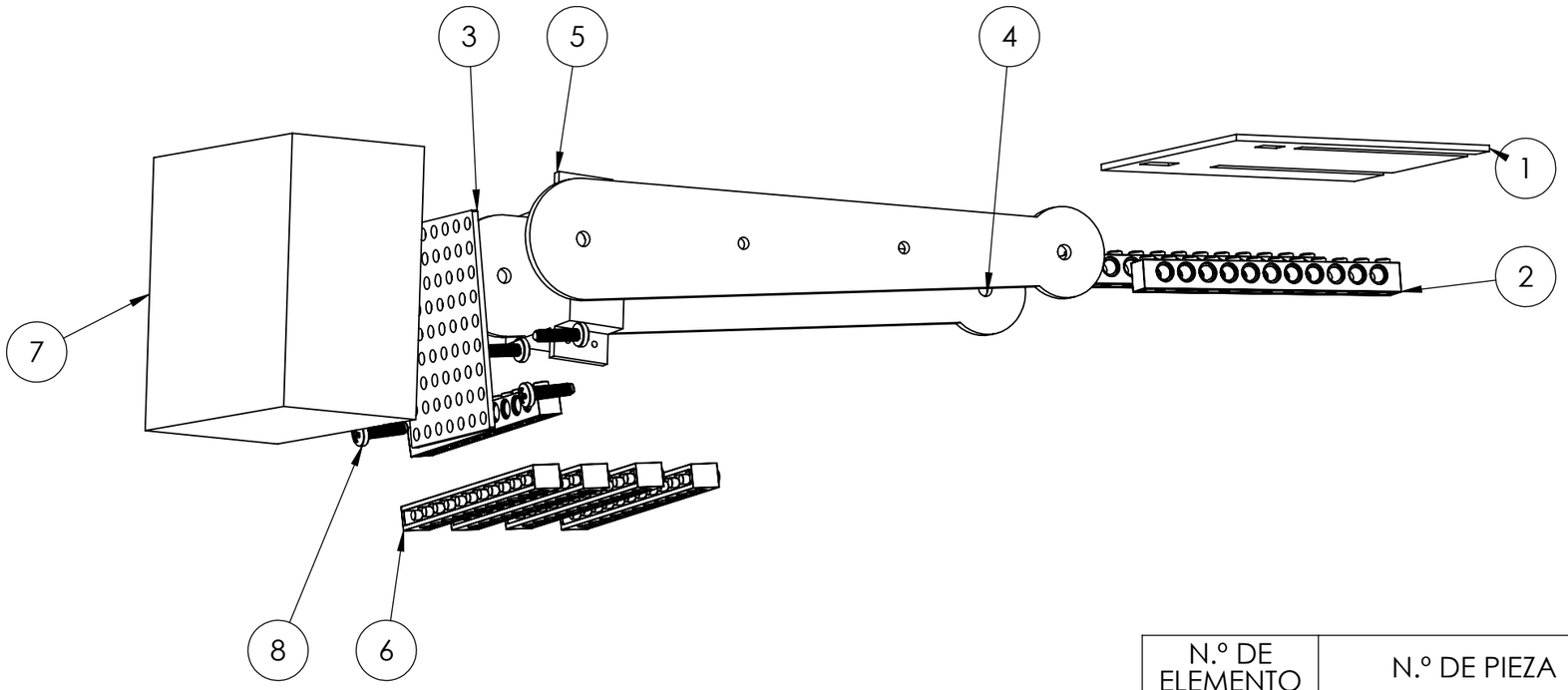
El total del proyecto asciende a 3997.49 € (TRES MIL NOVECIENTOS NOVENTA Y SIENTE EUROS CON CUARENTA Y NUEVE CENTIMOS)

Los precios de mano de obra han sido extrapolados de proyectos anteriores.

Los precios de los materiales se han obtenido del distribuidor: www.rs.es

El precio del Kit de brazo robótico se ha consultado en: https://www.amazon.es/

Los precios de los componentes de Lego Technic se han obtenido de: http://brickset.com/



N.º DE ELEMENTO	N.º DE PIEZA	DESCRIPCIÓN	CANTIDAD
1	Soporte STM	Sorporte de madera para la tarjeta STM32F3 Discovery	1
2	389521 Bright Red Technic Brick 1 x 12 with Holes	Piezas Lego Technic para el soporte	2
3	Plate 7x9	Plancha Make-Block 7x9	1
4	Piezas de Plástico	Piezas de plástico del kit brazo robótico	2
5	Motor DC	Motor DC del kit brazo robótico	1
6	Brick 1 x 12 with Holes	Brick Base Delantera	5
7	Contrapeso	Caja de contrapeso	1
8	Tornilleria	Tornillos M4x20	4

PROYECTO: NIVELADOR AUTOMÁTICO PARA TRABAJOS DE CONSTRUCCIÓN

Trabajo Fin de Grado

PROYECTISTA
Cerdá Andrés, Iván

Escuela Técnica Superior de Ingeniería del Diseño

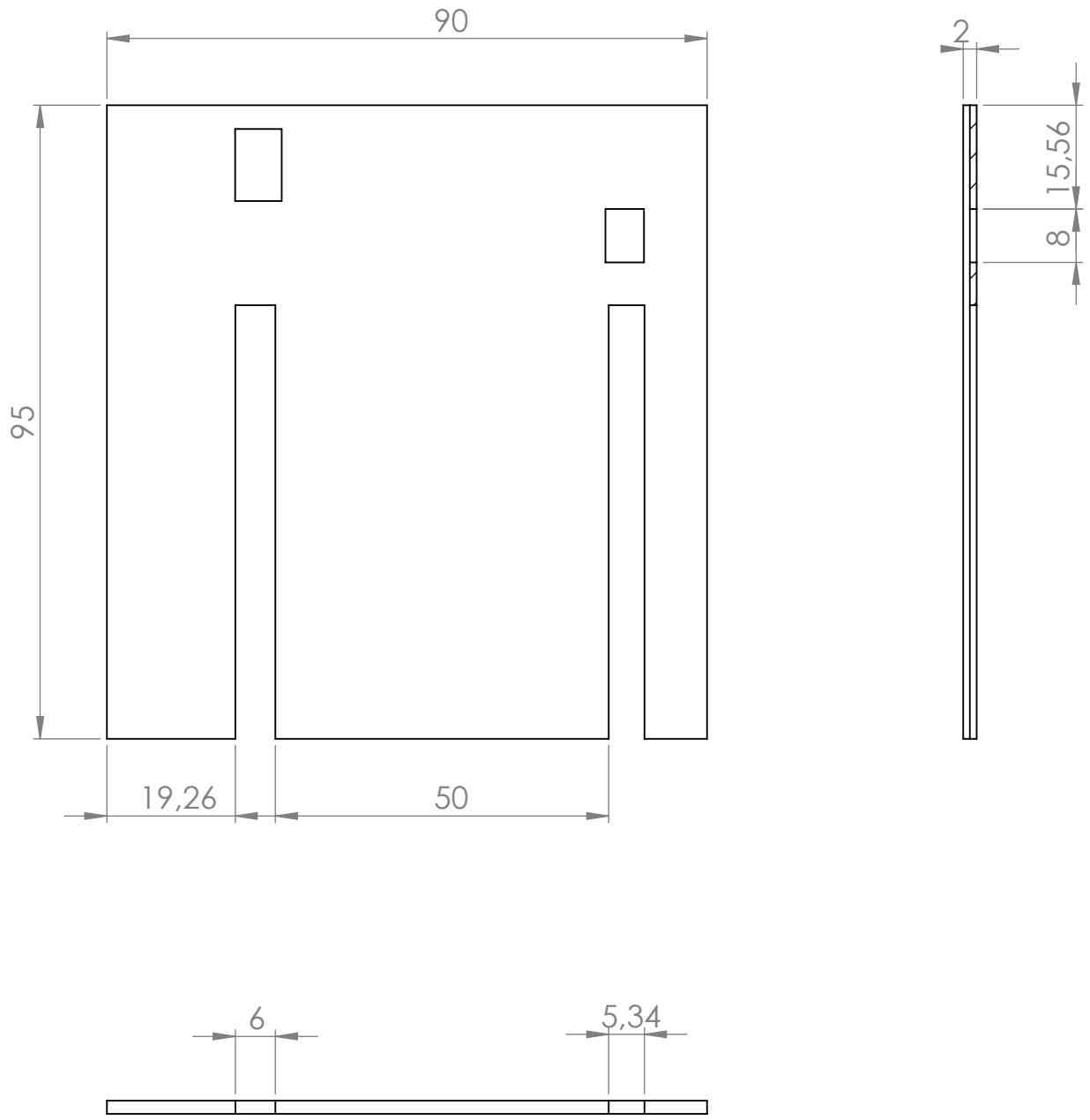
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

TÍTULO: Ensamblaje Prototipo

N.º DE DIBUJO Plano 1

ESCALA:1:2

HOJA 120 DE 122



PROYECTO: NIVELADOR AUTOMÁTICO PARA TRABAJOS DE CONSTRUCCIÓN

Trabajo Fin de Grado

PROYECTISTA

Cerdá Andrés, Iván

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

TÍTULO:

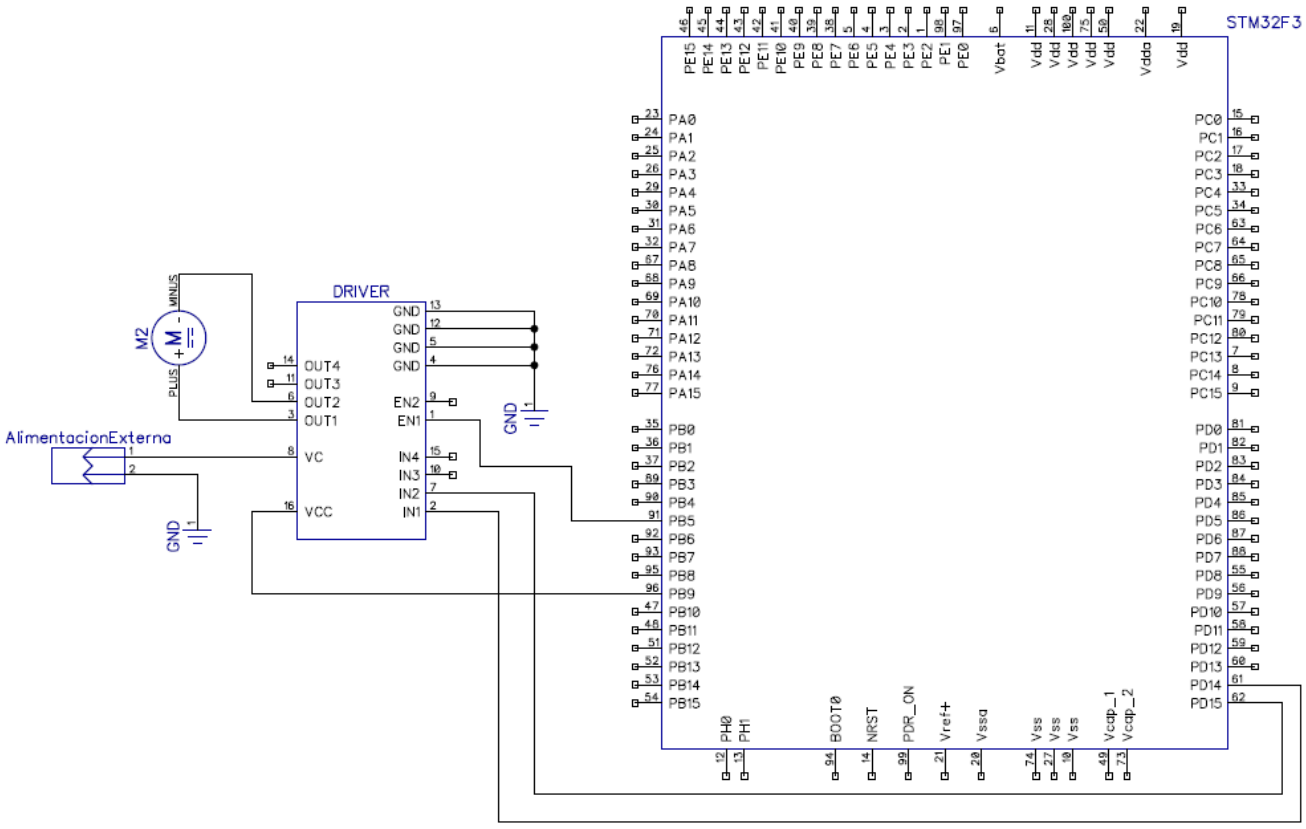
Soporte tarjeta STM

N.º DE DIBUJO

Plano 2

ESCALA:1:1

HOJA 121 DE 122



PROYECTO: NIVELADOR AUTOMÁTICO PARA TRABAJOS DE CONSTRUCCIÓN

Trabajo Fin de Grado

PROYECTISTA

Cerdá Andrés, Iván

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

TÍTULO:

Plano de conexiones

N.º DE DIBUJO

Plano 3

ESCALA:1:1

HOJA 122 DE 122